Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

2007-06

Mapping autonomous system's router level topology in IPv6

Poulin, Robert J.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/3412

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MAPPING AUTONOMOUS SYSTEM'S ROUTER LEVEL TOPOLOGY IN IPV6

by

Robert J. Poulin

June 2007

Thesis Advisor:	Geoffrey Xie
Co-Advisor:	John Gibson

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2007	Master's Thesis

4. TITLE AND SUBTITLE Mapping Autonomous System's Router Level Topology in IPv6	5. FUNDING NUMBERS
6. AUTHOR(S) Robert J. Poulin	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFGA 525 Brooks Road Rome, NY 13441-4505	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited	

13. ABSTRACT

The core of the Internet is composed of many independent and mutually exclusive collections of routers, called Autonomous Systems, which are responsible for moving traffic between communicating end-systems, or hosts, regardless of the relative location of those hosts. The complexity of the internal composition of theses autonomous systems is such that accurate documentation of their topology, reference to as mapping, is difficult and prone to error. Developing automated support for this effort remains an area of active research, the potential benefit of which is the ability to actively monitor the health of the Internet across these autonomous systems making it possible to identify critical infrastructure chokepoints before their failure adversely impacts the network or national security. The Internet is in the process of transitioning to a new version of the Internet Protocol, the fundamental protocol that melds the heterogeneous networks worldwide into a single cooperative whole. Tools, techniques, and tactics developed for the current version, IPv4, may hold promise for adaptation to support the new version, IPv6. This thesis explores several of the IPv4 techniques that hold promise for adaptation and provides an implementation as a proof-of-concept.

14. SUBJECT TERMS IPv6, Autonomous Systems, Mapping IPv6 Autonomous Systems, Anonymous Resolution, Alias Resolution, Doubletree, Rocketfuel, ATLAS, Router Level Topology	15. NUMBER OF PAGES 419
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

MAPPING AUTONOMOUS SYSTEM'S ROUTER LEVEL TOPOLOGY IN IPv6

Robert J. Poulin
Captain, United States Air Force
B.S., University of Nebraska Omaha, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2007

Author: Robert J. Poulin

Approved by: Geoffrey Xie
Thesis Advisor

John Gibson
Co-Advisor

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The core of the Internet is composed of many independent and mutually exclusive collections of routers, called Autonomous Systems, which are responsible for moving traffic between communicating end-systems, or hosts, regardless of the relative location of those hosts. The complexity of the internal composition of theses autonomous systems is such that accurate documentation of their topology, reference to as mapping, is difficult and prone to error. Developing automated support for this effort remains an area of active research, the potential benefit of which is the ability to actively monitor the health of the Internet across these autonomous systems making it possible to identify critical infrastructure chokepoints before their failure adversely impacts the network or national security. The Internet is in the process of transitioning to a new version of the Internet Protocol, the fundamental protocol that melds the heterogeneous networks worldwide into a single cooperative whole. Tools, techniques, and tactics developed for the current version, IPv4, may hold promise for adaptation to support the new version, IPv6. This thesis explores several of the IPv4 techniques that hold promise for adaptation and provides an implementation as a proof-of-concept.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION....1
A. OBJECTIVE....2
B. WHY IPV6?....2
C. RESEARCH QUESTIONS....3
D. OVERVIEW....4

II. BACKGROUND AND RELATED WORK....5
A. INTRODUCTION....5
B. DIFFERENT TYPES OF PROBING....6
C. DIFFERENT LEVELS OF MAPPING....6
D. PREVIOUS MAPPING EFFORTS....7
1. Mercator....7
2. Rocketfuel....9
3. ATLAS....12
4. DOUBLETREE....15
E. SUMMARY....17

III. LABORATORY NETWORK RESEARCH....19
A. INTRODUCTION....19
B. LABORATORY NETWORK....19
C. ALIAS RESOLUTION....21
1. CISCO IOS....21
2. JUNOS....24
3. Why the Difference in Responses?....25
D. ANONYMOUS RESOLUTION....26
1. Responses....26
2. Bisimilarity....27
E. SOURCE ROUTING....28
1. Capability Testing....28
2. Disabling Source Route....28
F. ADDRESS SPACE PROBLEM....28
G. ADDITIONAL NOTES....29

IV. DESIGN AND IMPLEMENTATION OF PROBING SYSTEM....31
A. INTRODUCTION....31
B. PROBING ENGINE....34
1. Main....37
2. Process Receive....38
a. Process Packet....38
b. Process Source....39
3. Checkpoint and Loader....48
4. Fill List....49
5. Done Check....49

6. Generate 49
7. Packet Generator 50
8. Packet Receiver 50
9. Probe Builder 51
10. Syslog 51
C. ALIAS RESOLVER 52
1. Checkpoint and Loader 52
2. Fill List 52
3. Process Alias Packet 53
4. Process Alias Source 53
5. Generate 53
6. Resolve 53
D. ANONYMOUS RESOLVER 55
1. Checkpoint and Loader 55
2. Resolver 55
E. CREATION OF INITIAL PROBES 56
1. Zebra-dump-parser 57
2. My_parse 57
a. Peerfile.txt 57
b. Allfile.txt 58
c. Asfile.txt 58
3. Sort 58
4. Traceroute6 58
5. Scapy 58
F. WRAP UP 59
V. SYSTEM SPECIFIC IMPLEMENTATION AND DEPLOYMENT 61
A. SYSTEM SPECIFIC IMPLEMENTATION 61
1. Top Level Directory 61
2. Directory - alias_test_files 62
3. Directory - anonymous_test_files 62
4. Directory - Main_test_files 63
5. Directory – src 64
6. Directory - stubs 66
7. Directory - Support Programs 67
B. DEPLOYMENT 67
1. Probing Engine 67
2. Alias Resolver 70
3. Anonymous Resolver 71
C. WRAP UP 72
VI. CONCLUSIONS, RECOMMENDATIONS AND FUTURE WORK 73
A. CONCLUSION 73
B. RECOMMENDATIONS AND FUTURE WORK 73
1. Probing Builder 74
2. Probing Engine 74
3. Alias Resolver 75

4. Anonymous Resolver 76

APPENDIX A. SCAPY SCRIPTS AND TEST BED CONFIGURATION 77

Test Case 1: 77

Test Case 2: 86

Test Case 3: 97

APPENDIX B. SOURCE CODE 109

ALIAS MAIN.C 109

ALIAS_MAIN.H 123

ALIAS_MAIN_HELPER.C 124

ALIAS_MAIN_HELPER.H 146

ALIAS_OFFLINE.C 147

ALIAS_OFFLINE.H 155

ANON_MAIN.C 156

ANON_MAIN.H 175

MAKEFILE 176

MY_PARSE.C 178

PROBE_BUILDER.C 184

PROBE_BUILDER.H 191

PROBE_CKPNT.C 192

PROBE_CKPNT.H 209

PROBE_EDGES.C 210

PROBE_EDGES.H 219

PROBE_GENERATOR.C 220

PROBE_GENERATOR.H 235

PROBE_GENERATOR_STUB.C 236

PROBE_LOADER.C 238

PROBE_LOADER.H 271

PROBE_MAIN.C 272

PROBE_MAIN.H 293

PROBE_MAIN_HELPER.C 297

PROBE_MAIN_HELPER.H 332

PROBE_MAIN_STUB.C 333

PROBE_RECEIVE.C 350

PROBE_RECEIVE.H 366

PROBE_RECEIVE_STUB.C 367

PROBE_STOP.C 374

PROBE_STOP.H 382

PROBE_UTILS.C 383

PROBE_UTILS.H 390

SYS_LOG.C 391

SYS_LOG.H 395

LIST OF REFERENCES 397

INITIAL DISTRIBUTION LIST 401

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1 – Topology Mapping. (From: (Donnet et al. 2006, 12)).....7
Figure 2 – Ingress Reduction (From: (Spring, Mahajan, and Wetherall 2002, 133-145)).....11
Figure 3 – Egress Reduction (From: (Spring, Mahajan, and Wetherall 2002, 133-145).....11
Figure 4 – Next Hop AS Reduction (From: (Spring, Mahajan, and Wetherall 2002, 133-145).....12
Figure 5 – Alias Resolution with source routing (From: (Waddington et al. 2003, 59-68)).....14
Figure 6 – Original Network (From: (Yao et al. 2003, 353-363)).....14
Figure 7 – Probed Network (From: (Yao et al. 2003, 353-363)).....15
Figure 8 – Local Stop Set (After: (Donnet et al. 2005, 327-338)).....16
Figure 9 – Global Stop Set (After: (Donnet et al. 2005, 327-338)).....17
Figure 10 – Lab Test Case 1.....21
Figure 11 – Lab Test Case 2.....24
Figure 12 – Bisimilarity (From: (Yao et al. 2003, 353-363).).....27
Figure 13 – Probing System Overview.....33
Figure 14 – Typical Probe.....35
Figure 15 – Probing Engine.....37
Figure 16 – Case A and Beginning of Case B.....41
Figure 17 – Case B.....42
Figure 18 – Case C.....45
Figure 19 – New Edge or Node.....47
Figure 20 – Finish with destination.....48
Figure 21 – Alias Resolver.....54
Figure 22 – Anonymous Resolver.....56
Figure 23 – Lab Test Case 1 Configuration.....85
Figure 24 – Lab Test Case 2 Configuration.....96
Figure 25 – Lab Test Case 3 Configuration.....107

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. – Contents of the top level directory....62
Table 2. – Contents of the alias_test_files directory....62
Table 3. – Contents of the anonymous_test_files directory....63
Table 4. – Contents of the Main_test_files directory....64
Table 5. – Contents of the src directory....66
Table 6. – Contents of the stubs directory....67
Table 7. – Contents of the Support Programs directory....67
Table 8. – Parameters used for probe engine....69

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This thesis is dedicated to my wife Dee and the children. During the entire process my wife always stood by my side and ensured the day to day things were done freeing me to work the endless hours required to complete this thesis and without her support none of this would have been possible. Dee thank you for everything you have done, this would not have possible without you. I would also like to thank my son Scott for understanding when I needed to leave Nebraska to obtain this degree and always supporting my work. Finally, I would like to thank Ashley and Courtney for understanding when they needed to leave their friends both in Nebraska and now in Monterey to follow my career.

I would like to thank my thesis adviser Geoffrey Xie for always making me look in a new direction when I ran into brick wall. I would also like to thank him for the continued push to move the thesis just a little further thus making this thesis into a reality.

I would like to thank my second reader John Gibson who was always willing to lend an ear and let me vent. He was also always willing to let me talk through my problems and help point me in the correct direction. Finally, I want to thank him for his support to make it through the rough times.

Next, I would like to thank Senior Lecturer Christopher Eagle for his help in writing all the code for this thesis. His insight and knowledge in programming languages are the reason all of the code came together and worked. His ability to quickly and clearly explain how processes worked in C made this all possible.

Finally, I would like to thank my sponsors, Air Force Research Laboratory and the National Security Agency. For without their donations and monetary support much of this research would not have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Mapping Internet topologies is important for many reasons and is the reason behind this research. To quote the developers of Rocketfuel, "Realistic Internet topologies are of considerable importance to network researchers. Topology influences the dynamics of routing protocols, the scalability of multicast, the efficiency of proposals for denial-of-service tracing and response, and other aspects of protocol performance (Spring, Mahajan, and Wetherall 2002, 133-145)."

To see an example in the world today one need look no further than a network administrator as she isolates the cause of a failure. Having an accurate map of a very large network is a must for these situations but this unfortunately is difficult and error prone. This arises because some administrators know their network through the use of manually maintained maps, unfortunately, when there are many administrators sharing responsibilities for a network, swapping new equipment for old, and moving less capable equipment to less utilized parts of the network, there exists the possibility of one of them not updating the map or incorrectly updating the configuration. Furthermore, if a user or someone unauthorized installs a router that they do not want the network administrators to know about, either for a legitimate or malicious purpose, the administrators may not find it until some significant event calls their attention to the device. Meanwhile, the router may open a back door or some other security vulnerability to their network.

These and other difficulties in maintaining accurate network maps can lead administrators to use automated network administration tools to track and discover the network's health and topology. These tools accomplish their feats through the use of network management protocols and injecting random traffic through the various routes. They also probe the network for routers and other devices that do not respond to management queries, either by accident or design. This probing helps the administrator find rogue or incorrectly configured devices. Using this information, administrators may

be able to correct potential security vulnerabilities before they are exploited and causes damage to the organization. Much research remains to be done in the area of automated support for mapping topologies.

A. OBJECTIVE

Specifically, this research focuses on the feasibility of porting IPv4 mapping methodologies to IPv6. It seeks to improve the performance of mapping methods by reducing the problem space for searching and mapping IPv6 Autonomous Systems by utilizing a small subset of selected addresses. Finally, this research ports one of the more successful distributed IPv4 topology discovery tools methodologies, "Rocketfuel" to IPv6 (Spring, Mahajan, and Wetherall 2002, 133-145), while incorporating the Doubletree method to reduce the probing load (Donnet et al. 2005, 327-338).

B. WHY IPV6?

IPv6 is the future of the Internet. IPv6's primary reason for development is to deal with the lack of address space in IPv4. The address space in IPv4 is limited to approximately 4 billion specific addresses. Due to the Internet's explosive growth, as early as the1990's it was recognized that the address space needed was much larger than what was available. Many methods were introduced to delay the exhaustion to include network address translation and Classless Inter-Domain Routing (CIDR), significantly delaying the eventual exhaustion of addresses. Unfortunately, none solved the problem. Current predictions still show the address space running out sometime between 2009 and 2041 depending on which growth model is used, (Popoviciu, Levy-Abegnoli, and Grossetete 2006). Even if the address space lasts for another 200 years it will become a lengthy and difficult process to obtain an IP address as the remaining space dwindles, increasing the burden for everyone (Malone and Murphy 2005). The only true solution to the problem for the long term is to adopt a severe address philosophy change, as included in IPv6 (Malone and Murphy 2005).

Of course, IPv6 also provides some other benefits. First, by using the lessons learned with address aggregation in IPv4, the implementation of IPv6 seeks to avoid the

explosive routing table growth at the core of the Internet that occurred with IPv4. The average routing table more than doubled, from 34,000 to 76,000 routes, in the six years following 1994 (Popoviciu, Levy-Abegnoli, and Grossetete 2006). IPv6 is built for speed, by taking many of the complicated problems that routers must handle in IPv4 and moving them to the end-hosts. Following this philosophy, routers will no longer fragment but instead will send an error message when a packet exceeds the supported size of the least capable link traversed, thus shifting the load to the end-hosts to fragment the packets. Simplifications like this, allows the routers to take most of the remaining work and shift it to hardware, increasing their speed without having to worry about dealing with special cases. Third, auto-configuration will prove to be a huge benefit of IPv6. With IPv4, when you "plug" into a network the Dynamic Host Configuration Protocol (DHCP) can configure your system with an appropriate IP address and other information, as preconfigured on the DHCP server by the network administrator. IPv6 provides an additional capability, fully automating the host configuration. This auto-configuration allows for router renumbering, providing enhanced support for mobile networks, not just mobile hosts. This means a network may be as mobile as the users being supported. All of the systems can maintain the same last 64 bits and the first 64 bits can be automatically set by the router based on where it is plugged into the Internet (Malone and Murphy 2005), greatly advancing adaptability, a crucial characteristic for military networks. Security implications pertinent to these enhanced capabilities must be considered when deployment decisions are made, but are beyond the scope of this thesis.

Automated topology mapping of IPv6 networks may benefit from research already conducted for IPv4 networks which is where this thesis begins. The following issues must be explored to determine whether the techniques and tactics employed for interrogating IPv4 networks hold promise for application with IPv6-based networks

C. RESEARCH QUESTIONS

1. Is source routing still a viable topology discovery method in an IPv6 environment?
2. What methods for mapping router-level topology will port from IPv4 to IPv6?

3. Are there new ways to reduce the size of the address space for locating routers and their interfaces? Does the address space matter?
4. Are there new methods of detecting alias and anonymous routers to reduce the graphs produced by a topology mapping tool.
5. Can the techniques used in “Rocketfuel” be ported to IPv6?

D. OVERVIEW

This thesis will give research background into many of the methods used today in IPv4 and explain their application to IPv6. Then it will cover the results ob laboratory experiments and how they will apply to the probing system. Next, it will cover the implementation of a system to automatically map an IPv6 network and the specific tools required to accomplish the task. Finally, it will detail the results of operating the system on a live IPv6 network and detail future improvements.

II. BACKGROUND AND RELATED WORK

A. INTRODUCTION

IPv6 has completely revamped the IP header from IPv4. Some of the major differences are the lack of any of the fields for fragmentation. Packets too large for the network are rejected and an error message is returned to the sender thus pushing the job away from the core and back to the edges. The "type of service" field is replaced with the flow label field which is designed for quality of service (Popoviciu, Levy-Abegnoli, and Grossetete 2006). IPv6 addressing has also changed: broadcast addresses are gone and replaced by scoped multicast addresses and there is also the introduction of the anycast address. "Anycast addresses are assigned to many machines but each packet is delivered to only one of those machines (Malone and Murphy 2005)". There are numerous other differences, too many to list here, Malone provides a nice introduction (Malone and Murphy 2005).

Due to these and other changes in the protocol, routers and their operating systems have been redesigned to handle the differences. This means previous methods of mapping networks in the IPv4 Internet may not port to IPv6. For example, any method using the identification field in the IPv4 header will not port since that field is deprecated in IPv6. This chapter will examine existing IPv4 network mapping techniques and detail what will not port and what may port to IPv6 and then introduce some possible new concepts. Finally, this thesis will deal with topology maps created via a black box approach. A black box approach means that only information available or deducible by the general public will be used to create the maps.

This chapter begins by defining different types of probing techniques using either a single source or a set of distributed sources. Next, Internet topology mapping schemes are explained and some of the specifics on the requirements of each are covered. Finally, previous efforts in the area of Internet mapping are detailed and areas influential in the completion of this thesis are discussed.

B. DIFFERENT TYPES OF PROBING

There are essentially two different ways to probe networks. The first is single source or limited source probing; this is essentially where you use a single point or a limited number of points. Limited in this context means a number and spread that leads to a significant number of probed end-points being more than 10 hops away from the probing source. This probing method has the advantage of not requiring a lot of infrastructure, which makes it convenient for researchers. Probing of this type was used by early efforts to map the Internet but was found to produce inaccurate maps due to spatial bias. Spatial bias is where the degree distribution of the probed network depends on the distance of the nodes from the probing source and not the actual mapped network (Donnet et al. 2005, 327-338). Overcoming spatial bias is done by probing from a large number of distributed sources and targeting the end points in an informed manner, thus allowing the probe lengths to remain short and ensuring probes have good distribution around the target (Donnet et al. 2005, 327-338). For more information on this subject see Clauset and Moore's paper in the list of references.

C. DIFFERENT LEVELS OF MAPPING

There are essentially three levels of Internet topology that can be mapped. The first is at the interface level and would be depicted as the dark black dots in the circles of Figure 1. The second is at the router level and is depicted as the white circles in the same figure. In order to make this map you have to combine all the interfaces of a router to the single router during the mapping process. Finally, there is the Autonomous System (AS) level of mapping, which maps the links between each cloud of the figure under the assumption that each cloud is controlled by a single administrating entity (Donnet et al. 2006, 12).

Router level mapping subsumes a subtype confined to mapping within a single AS. This thesis will deal with this specific sub-area. This sub-area was chosen because it is practical to system administrators when dealing with the area of reachability and not subject to spatial bias if done correctly.

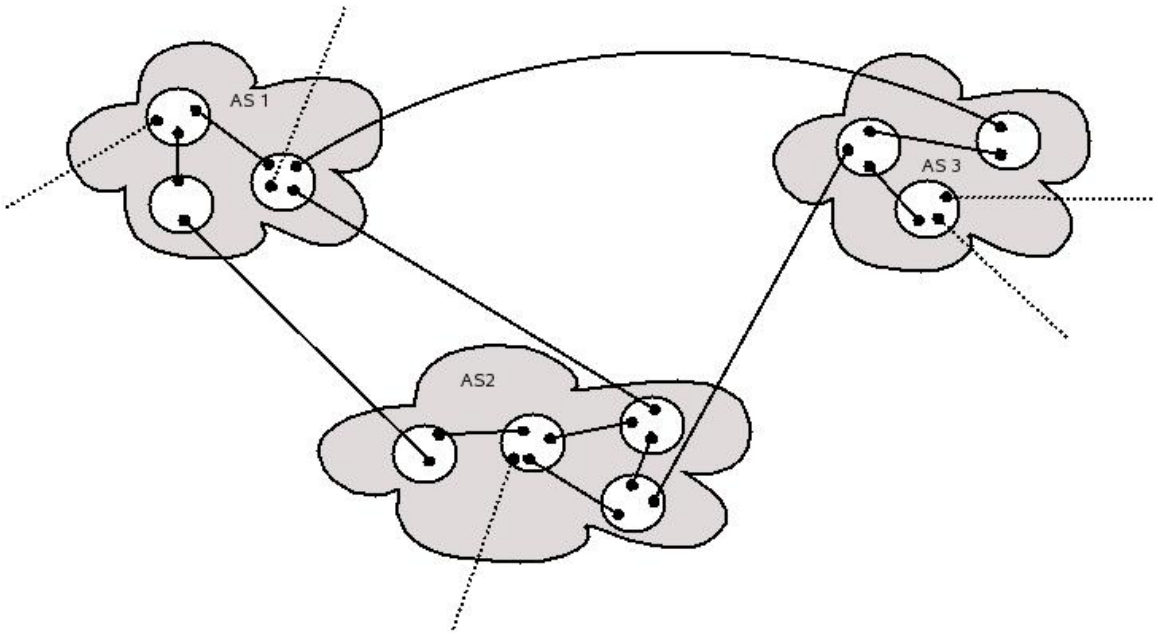

Figure 1 – Topology Mapping. (From: (Donnet et al. 2006, 12))

D. PREVIOUS MAPPING EFFORTS

In this section, previous mapping efforts are reviewed in their order of completion. For brevity, only major contributions that are related to this thesis are discussed.

1. Mercator

This paper is cited by almost every paper in this field. While some of the methods used in this paper are beginning to show their age, there are still many in use today. Their entire effort, based on network mapping at the router level, yielded three areas of importance to this research.

The first contribution is source routing. Source routing is an IP routing option that allows you dictate the (complete or partial) sequence of hops a packet should traverse in order to reach its destination. The ordered list of hop or destination identifiers are encoded as an optional part of the packet's IP header, and upon reaching each destination the router then swaps the regular destination field of the IP header with the next destination in the list and then routes the packet to that next destination. The Mercator

work shows that provided the routers are distributed properly, even if only five percent of routers are source routing capable, one can effectively map 90 percent of a network (Govindan and Tangmunarunkit 2000, 1371-1380). Each source routing capable router is like having a separate probing engine running from that location (Govindan and Tangmunarunkit 2000, 1371-1380). Furthermore, this paper presents a method of detecting source routing capable routers, in which a UDP packet is sent to a high port destined for destination A with a source routing header included for destination B. If A responds with a port unreachable control message then the router is not source route capable; if B responds then A is source route capable (Govindan and Tangmunarunkit 2000, 1371-1380). This thesis will test this methodology to see if it still works in IPv6.

The second method of interest reduces the probing load by starting each subsequent probe at a distance beyond the distance already mapped. By doing this they do not map the first few hops repeatedly with each new destination. This method was much improved by the Doubletree effort (Govindan and Tangmunarunkit 2000, 1371-1380), as described later in this chapter.

The final contribution of Mercator is in the area of alias resolution. Alias resolution exists because, "(p)ath probes actually discover router interfaces (Govindan and Tangmunarunkit 2000, 1371-1380)." This means that a set of probes, "might discover more than one interface belonging to the same router (i.e., multiple aliases for the same router) (Govindan and Tangmunarunkit 2000, 1371-1380)." A requirement of any probing engine mapping router level topology is the ability to resolve these interfaces back to one hosting router. The Mercator project proposed a scheme to meet this requirement by leveraging the fact that a router's response to a UDP probe to an unnumbered port through any of its interfaces always contains the address of the outgoing interface of the router (Govindan and Tangmunarunkit 2000, 1371-1380). It should be noted that this router behavior is not part of the IPv4 specification while it appeared to be implemented by all the routers probed by Mercator.

2. Rocketfuel

Rocketfuel was one of the most well known and successful attempts for mapping the router level topology of an AS. This effort was so successful that Rocketfuel became one of the prime engines in the test bed on the Planetlab network (Spring, Wetherall, and Anderson 2003).

The first major contribution of Rocketfuel was successfully implementing a method of distributed directed probing that all but eliminated spatial bias. Distributed directed probing allowed Rocketfuel to choose end points in an informed manner that surrounded the AS and ensured every point in the As was approached from a different entry point. After thorough analysis by later researchers this method was proven to all but eliminate spatial bias (Clauset and Moore 2003).

The second major contribution was its ability to reduce the number of probes needed for the targeted network without compromising too much accuracy of the resultant map. To reduce their probing load they used data from the RouteViews project to target their probes (Meyer 2007). This data consists of Border Gateway Protocol (BGP) table entries that show the sequence of ASes a packet would traverse on its way to a destination prefix from a given prefix. Rocketfuel first picked sources and destinations that would include the targeted AS in the path to form its initial pool of directed probes (Spring, Mahajan, and Wetherall 2002, 133-145). Second, it performed ingress reduction. Ingress reduction is where one probe is eliminated if two probes originating from two different ASes enter and exit the targeted AS at the same pair of ingress/egress points as in Figure 2 (Spring, Mahajan, and Wetherall 2002, 133-145). Third, it performed egress reduction. Egress reduction eliminates one probe if two destination prefixes use the same ingress and egress routers, as in Figure 3 (Spring, Mahajan, and Wetherall 2002, 133-145). The final reduction method used was next-hop AS reduction, by which if two probes destined to different prefixes share the same next hop AS, as in Figure 4, only one is chosen (Spring, Mahajan, and Wetherall 2002, 133-145). The total probe reduction

from all of these efforts was approximately 99 percent, while still maintaining above 90 percent accuracy on access routers and 65 percent on backbone routers (Spring, Mahajan, and Wetherall 2002, 133-145).

The third major contribution, particularly relevant to this thesis, was in the area of alias resolution. Rocketfuel provided clever methods to increase the accuracy of alias resolution. The first was the use of the TTL field to determine candidates for alias resolution (Spring, Mahajan, and Wetherall 2002, 133-145). The second was using the identification field of the IP header (Spring, Mahajan, and Wetherall 2002, 133-145). While this method was extremely valuable and accurate it is not valid in IPv6 since the identification field, used for packet fragmentation/reassembly, does not exists in the IPv6 header. The third method they developed is still extremely useful and is called rate limiting. Rate limiting sends enough probes to one of the alias addresses to cause the router to begin rate limiting the number of ICMP messages sent, at which time a probe is sent to the other address to see if the router responds (Spring, Mahajan, and Wetherall 2002, 133-145). Next, probes are sent to the addresses in reverse order and if the probe to the second address is again unanswered then the two addresses have a high likelihood of belonging to the same router (Spring, Mahajan, and Wetherall 2002, 133-145). While this method is valuable it has to be used sparingly because you are basically performing a denial of service on the router and this could cause an angry response by a system administrator. Furthermore, IPv6 relies on ICMP and this could cause problems for regular users of the network, especially in the area of path MTU discovery.

Finally, Rocketfuel attacked the problem of deducing to which AS a router belongs. Since most ISPs are unwilling to release maps of their networks for security reasons, determining if a router belongs to the ISP is a difficult process. NOTE: An ISP may control one or more ASes. Rocketfuel tackled the problem first by using subnets. For example, if an AS controlled a /24 subnet then it controlled all the /25 subnets below it (Spring, Mahajan, and Wetherall 2002, 133-145). Then they furthered the process through the use of reverse Domain Name System (DNS) requests (Spring, Mahajan, and Wetherall 2002, 133-145). Based on the naming of routers they were able to identify routers that belong to the ISP and also eliminate routers that belong to customers of the

ISP (Spring, Mahajan, and Wetherall 2002, 133-145). Finally, using these names, they were very often able to determine the location and further help determine the purpose of the router and its role in the network. This information then allowed them to discover more parts of the network (Spring, Mahajan, and Wetherall 2002, 133-145).

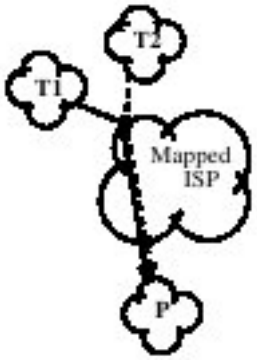

Figure 2 – Ingress Reduction (From: (Spring, Mahajan, and Wetherall 2002, 133-145))

Figure 3 – Egress Reduction (From: (Spring, Mahajan, and Wetherall 2002, 133-145)

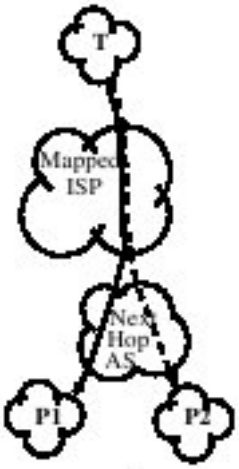

Figure 4 – Next Hop AS Reduction (From: (Spring, Mahajan, and Wetherall 2002, 133-145)

3. ATLAS

This is the only true effort I have seen to map the router level topology of an IPv6 network. This effort was conducted on the 6Bone network, which as of the writing, of this thesis has been shut down, unfortunately. Still this effort was important, as it appears to be the first of its kind. The effort was conducted using a single source for discovery and probing with source routed probes, as suggested in the Mercator paper, to mitigate spatial bias (Waddington et al. 2003, 59-68).

Atlas' first important contribution was a unique method of alias resolution using source routed probes. The method counts on the fact that the source routing header is processed first by the routers before the hop limit is checked. Using this assumption, as shown in Figure 5, given a probe from P, if X and Y are different interfaces for the same router, then depending on the direction of the probing, different results will be obtained. If, for example, we probe Y using a UDP packet to a high numbered port from the X side then the router will first swap the destination address X and replace it with Y and then check the hop limit (Waddington et al. 2003, 59-68). If the hop limit is 1 then we will receive a hop limit exceeded message with a source address of Y (Waddington et al. 2003, 59-68). When the same probe is sent with the hop limit increased by 1 we will

receive a port unreachable message with a source address of Y (Waddington et al. 2003, 59-68). Repeating this experiment coming from the Y side we will receive the messages with a source address of X (Waddington et al. 2003, 59-68). The only check that remains is to ensure that the hop limits on the return packets are the same for both responses to ensure we don't hit the special case where there are two distinct routers and the outbound interface is anonymous on both (Waddington et al. 2003, 59-68). Note here, though, this is based on the assumption that routers will use the destination address as the source when replying from an anonymous interface. The assumption may not hold for today's routers and will need to be checked through experiments. Anonymous interfaces are interfaces configured with no global address.

The most significant related work, from the perspective for this thesis, was not Atlas itself but a paper they published while developing Atlas. That paper dealt with mapping networks consisting of anonymous routers (Yao et al. 2003, 353-363). An anonymous router is a router that either does not send a response back to a probe or uses the destination address of the probe as the source address for the response (Yao et al. 2003, 353-363). The router is anonymous because you know it is there due to the lack of response at that hop count but you have no way to uniquely identify it. For example, consider the network in Figure 6 where the circles are normal nodes answering with their address and the squares are anonymous nodes (Yao et al. 2003, 353-363). If you were to map the network Figure 6 using traceroute-like probes your map would look the one in Figure 7 (Yao et al. 2003, 353-363) with the squares containing the $\perp$ symbol representing the anonymous nodes 7 and 8 in Figure 6. Furthermore, the "topology inference optimization problem is NP-complete and approximating it within n^{∂} (where n is the size of the probe result, and ∂ a fixed constant) is NP-hard (Yao et al. 2003, 353-363)". This means there is no perfect way to deduce a 100 percent accurate map in polynomial time in the presence of anonymous routers. Thus, we must use imperfect heuristics or avoid the problem all together. Yao's paper deduced some heuristics for this purpose (Yao et al. 2003, 353-363).

Finally, one of the shortfalls of Atlas was the fact they avoided the issue of address space and conducted exhaustive address probing and, as the authors admit, this

will not be a viable method on a real IPv6 network if you want to finish in your lifetime (Waddington et al. 2003, 59-68). The address space problem has to do with the fact that the smallest subnet in IPv6 has 2^{64} addresses and at even 1 millions probes per second, generating 400 Mbps of traffic, on a subnet with 10,000 hosts it would take 28 years to find the first host if the hosts were spread out on the subnet address wise equally (Convery and Miller 2006, 43).

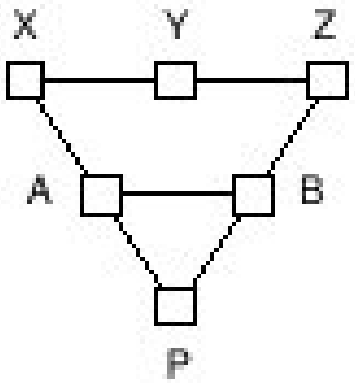

Figure 5 – Alias Resolution with source routing (From: (Waddington et al. 2003, 59-68))

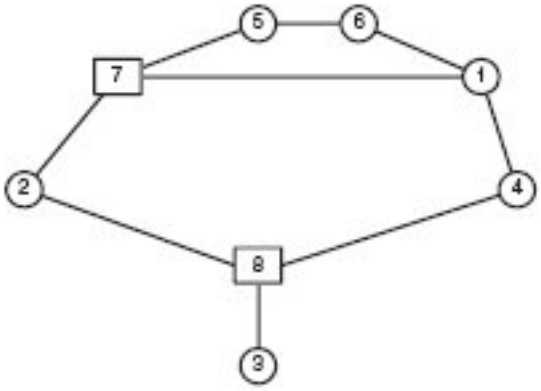

Figure 6 – Original Network (From: (Yao et al. 2003, 353-363))

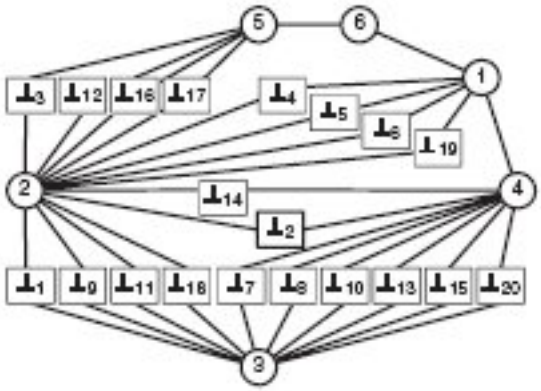

Figure 7 – Probed Network (From: (Yao et al. 2003, 353-363))

4. DOUBLETREE

This effort was the only relevant effort I found that took a hypothesis that many held to be true but could not prove, validated the concept, and turned the concept into a working model. One of the problems with distributed probing is overloading the network, especially when mapping a particular node from multiple destinations, which is required to avoid spatial bias. If you have a distributed mapping engine the actions of the engine could look to a distant end-point like a Distributed Denial of Service (DDOS) attack (Donnet et al. 2005, 327-338). For example, Skitter, a well known engine that maps intra-AS links, on the average locates 131K unique interfaces but to do this it averages about 54 million probes (Donnet et al. 2005, 327-338). Doubletree on the other hand, for the same number of interfaces manages to reduce the load to between 13 million and 32 million probes while maintaining accuracy above 90 percent (Donnet et al. 2005, 327-338). This reduction is the single largest accomplishment of Doubletree and where it contributed the most to this research.

It accomplishes the reduction through the use of stop sets. The first set is the local stop set, as shown in Figure 8, where in a probing engine sends out probes to a number of destinations for which it knows that the addresses up to a point, the large black dot, will be the same for all destinations (Donnet et al. 2005, 327-338). The second set is a global stop set, shown in Figure 9, where all probes from distributed locations to the same destination converge through an identical point, so probing beyond that point is fruitless (Donnet et al. 2005, 327-338). To use this knowledge, Doubletree starts from some point Z and probes backwards, away from the destination, until it reaches the source or a common point local stop set element, (Figure 8) hit by a previous probe from that engine (Donnet et al. 2005, 327-338). Then Doubletree probes from point Z until it either reaches the destination or one of the elements of the global stop set, (Figure 9), hit by another probing engine (Donnet et al. 2005, 327-338). The key to the technique is the choice of point Z; too close to the source and they found it would effectively do nothing except reduce accuracy; too far and it would still look like a DDOS to the destination (Donnet et al. 2005, 327-338).

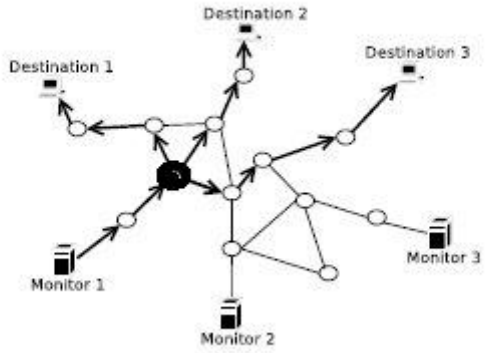

Figure 8 – Local Stop Set (After: (Donnet et al. 2005, 327-338))

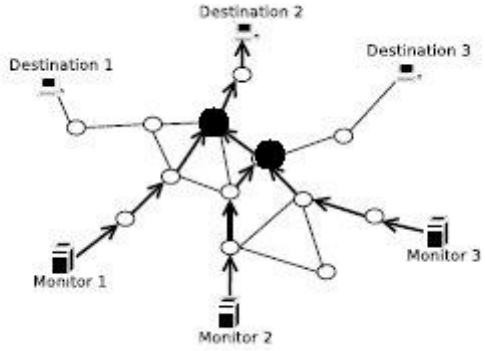

Figure 9 – Global Stop Set (After: (Donnet et al. 2005, 327-338))

E. SUMMARY

The intention of all the mapping techniques both Doubletree and Rocketfuel and combine them into one engine for mapping a live IPv6 AS. As for the resolution techniques and source routine the intention is to take their methods and experiment with them in a lab environment and see how they can be applied in the final product. The laboratory work is focus of the next chapter and the chapters after that will cover implementation and deployment of all these methodologies.

THIS PAGE INTENTIONALLY LEFT BLANK

III. LABORATORY NETWORK RESEARCH

A. INTRODUCTION

The purposes of the laboratory tests were three-fold: (i) test old methods and develop new techniques of alias resolution and anonymous resolution, (ii) test source routing, and (iii) determine the scale of the address space problem. The goal of all the tests in this chapter was to identify a set of effective methods to be integrated into a single multi-stage probing engine that is capable of mapping a live IPv6 network.

In development of the probing engine several key capabilities were identified as essential to make the probing engine a reality. The first was the ability to recursively locate the edges inside the AS with the minimum amount or probes possible using the Doubletree method. The design of this portion of the engine is explained in the next chapter of the thesis due the fact that we lacked the equipment to build anything more than a trivial mock up for the probing engine. However, the following capabilities required some initial laboratory testing to develop functional methods for an IPv6 network. The second key capability required was alias and anonymous resolution: if there was no method of making these functions happen then any graph generated would be highly inaccurate. Third was the capability to perform source-routing. As the facilities to perform distributed probing in IPv6 did not exist as of this writing, the ability to test and locate source routing capable nodes was extremely important to avoid the effects of spatial bias on the resultant maps. Finally, the engine needed to deal with the address space problem to produce a map of any usefulness.

B. LABORATORY NETWORK

The basic lab setup is shown in Figure 10. This setup was chosen for several reasons. First was to ensure we could put the router Merlot in a situation where it could still function but not have a global unicast address. Second, was to create multi-hop links between hosts. Finally was to give us an opportunity to test the functionality of source routing. All tests were performed on a mix of CISCO 2600 series and CISCO 3845

routers, a Juniper J4300 router, and a Juniper M7i router. The CISCO routers were running versions 12.3(15b) and 12.4(11)T with the advance IP services of CISCO's IOS. The Juniper routers were running a mix of 8.2R1.7 and 7.5R1.12 versions of JUNOS. All routers were returned to the factory original settings by the command "write erase" or "load factory-default" for IOS and JUNOS, respectively. All routing was done by the RIPng protocol; no other routing protocol was used since mapping only deals with reachability. Since the path taken by an individual packet does not change by the type of routing protocol used but by the policy implemented using that protocol, RIPng was chosen for simplicity of set up. While different standard manufacturer settings on the routers were experimented with, no firewall rules were ever activated, on since this could be highly individual for each site and possibly each router. Next, routers Merlot and Zinfandel were replaced with a Juniper J4300 and M7i respectively for the Juniper tests and all the test cases were repeated. All hosts were running SUSE Linux 10.1 with kernel 2.6.16.13-4-smp. All probes were sent from host 4 and then duplicated again from the other hosts to ensure consistency. For discussions in this thesis, host 4 will always be the source, since it was found that the source did not matter during these tests.

All tests were performed using a tool called Scapy that allows you to craft packets as needed using Python (BIONDI 2007). Numerous tests were conducted and the scripts/configurations for those with significant results are included in Appendix and numbered according to their test case.

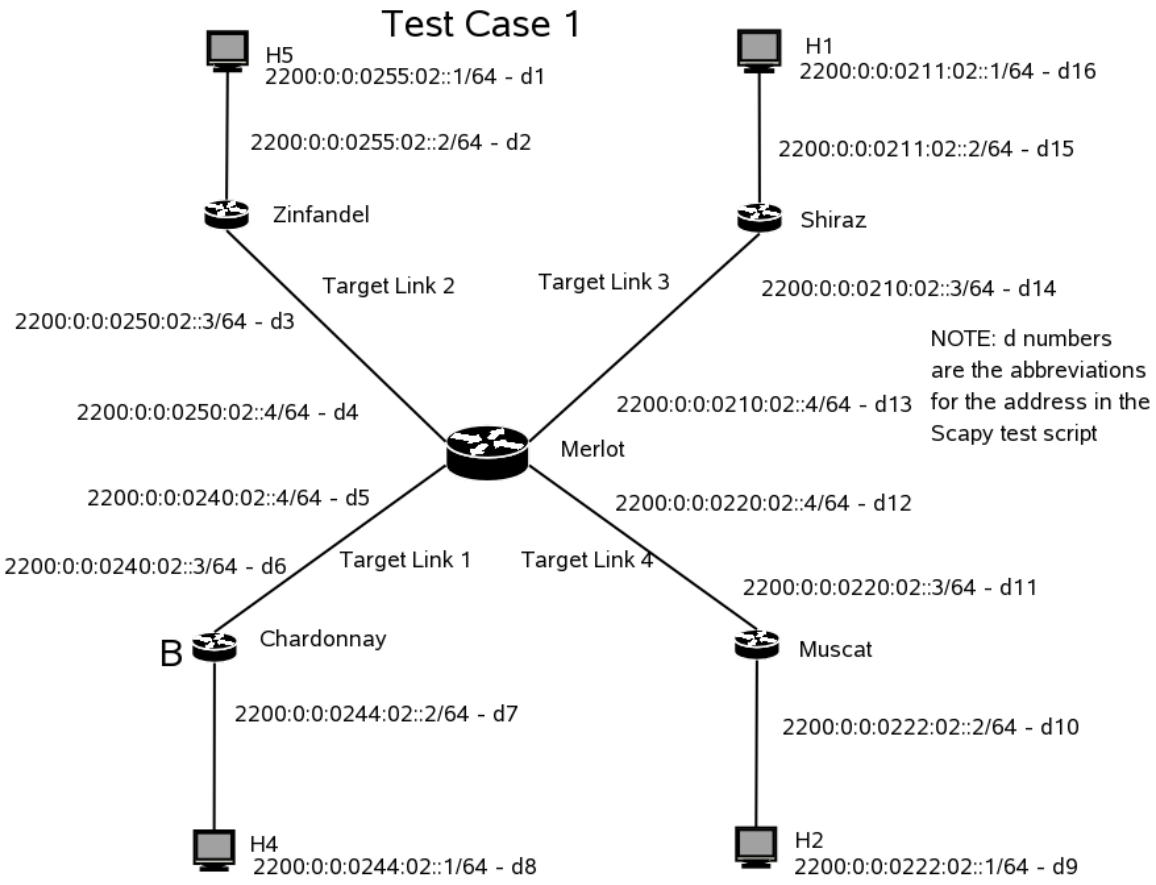

Figure 10 – Lab Test Case 1

C. ALIAS RESOLUTION

As explained previously, alias resolution is an attempt to reduce all responding interfaces of a router to one router. During the lab tests, significant behavior differences between IOS and JUNOS were noted.

1. CISCO IOS

Starting with UDP, it was noted during experiments that IOS responded to UDP probes with ICMPv6 messages using the receiving interface's address as the source IPv6 address. This behavior did not vary if the probe was directed to another interface on that router or if the response was due to an expired hop limit. For example, in Figure 10, a UDP probe to interface d13 of router Merlot from host 4 would receive an ICMPv6

message with interface d5 as the source. Furthermore, the same source would be used for hop-limit-expired messages that traversed the router Merlot, regardless of the upper level protocol.

Next, both ICMPv6 and TCP, were acted upon in a similar fashion by IOS, so they will be considered together. It was observed that CISCO routers responded using the address associated with the interface pinged with either the ICMPv6 echo request or the TCP SYN as the source address of the response. For example, in Figure 10, an ICMPv6 ping to d13 resulted in an ICMPv6 reply with the address of d13 as the IPv6 source address. Apparently, CISCO chose to follow rule 2.2(b) of RFC 4443 for UDP packets and rule 2.2(a) for ICMPv6 and TCP (Conta and Deering 2006, 23). The author holds that this position allows for the most backwards compatibility, given the fact that Cisco routers were found to respond similarly in IPv4 (Govindan and Tangmunarunkit 2000, 1371-1380).

There were exceptions to the findings above, specific to IPv6. If the outgoing interface did not have a global unicast address as in Figure 11 or if the unnumbered command was used then one of the two following rules were used. "The ip(v6) unnumbered configuration command allows you to enable IP processing on a(n) … interface without assigning it an explicit IP address.(Anonymous)" First, if the interface was programmed with the unnumbered command it would use the address of the interface specified with the unnumbered command. This applied even if the interface had a global unicast address. So, if a probe was sent to interface d13 through interface d5 and d5 was configured with the unnumbered command specifying the IPv6 address of interface d12 as its response address then the router would reply to UDP probes with the IPv6 address of d12 as it source. There were no tests conducted to see the response to a probe to interface d5 if it had a global unicast address and the unnumbered command was specified for the interface due to the fact that black box mapping would have no way of obtaining that global unicast address. Next, if the interface had no global address and the unnumbered command was not used then CISCO routers chose an interface with a global unicast address to use as the source address of the reply. The choice of interface to use seemed to be the first one programmed on the same network module card as the interface

without a global unicast address, but this was not confirmed, since the importance for this thesis was that it would respond the same way every time. For example, if an UDP probe was sent to interface d13 through interface d5 and d5 had no global unicast address and the unnumbered command was not used then Merlot would use the address of interface d12 or interface d4 depending on which was programmed first.

Finally, in release 12.4 of CISCO IOS, CISCO added the additional ICMPv6 command called "ICMPv6 unreachable". This command suppresses port unreachable error messages generated by the interface for which it is configured. This command was found to be another inconsistent implementation by CISCO. For example, probing interface d13 through d5 in Figure 10 with a UDP probe and having interface d5 configured with the "no ICMPv6 unreachable" command meant there would be no response; likewise a probe to interface d5 would not be answered. The strange part came when the configuration was swapped around and interface d13 was configured with "no ICMPv6 unreachable" but interface d5 did not have that command, then port unreachable responses were received by probes through d5 to interface d13 or d5. Again, it is believed that this happened because of CISCO's choice of implementation of ICMPv6, as explained earlier. Finally, when both interfaces were configured with the "no ICMPv6 unreachable" command then neither responded with port unreachable. Also of interest, the router responded with time exceeded messages even if the "no ICMPv6 unreachable" command was specified on all of the interfaces.

To conclude, alias resolution should be possible on CISCO routers since they respond with different addresses depending on the type of packet they receive and the route the packet traverses to reach the router. This should allow the reduction of all of a router's interfaces down to a specific router entity through the use of ICMPv6 and UDP probes from different sources. The only exception expected will be if the ICMPv6 unreachable command is set on all the interfaces. For example, using Figure 10 as our test network, if we wish perform alias resolution on the router Merlot we would proceed as follows. Assume we have received ICMPv6 time exceeded messages from all of its interfaces (d4, d5, d12 and d13) by probing from all the hosts surrounding the router. If we then probe each interface address in succession with an UDP probe, lets say in order

from d4 through d12, they would each return the address of d5 and we could replace each address d4, d12, and d13 with the address of d5 and create one router in our map.

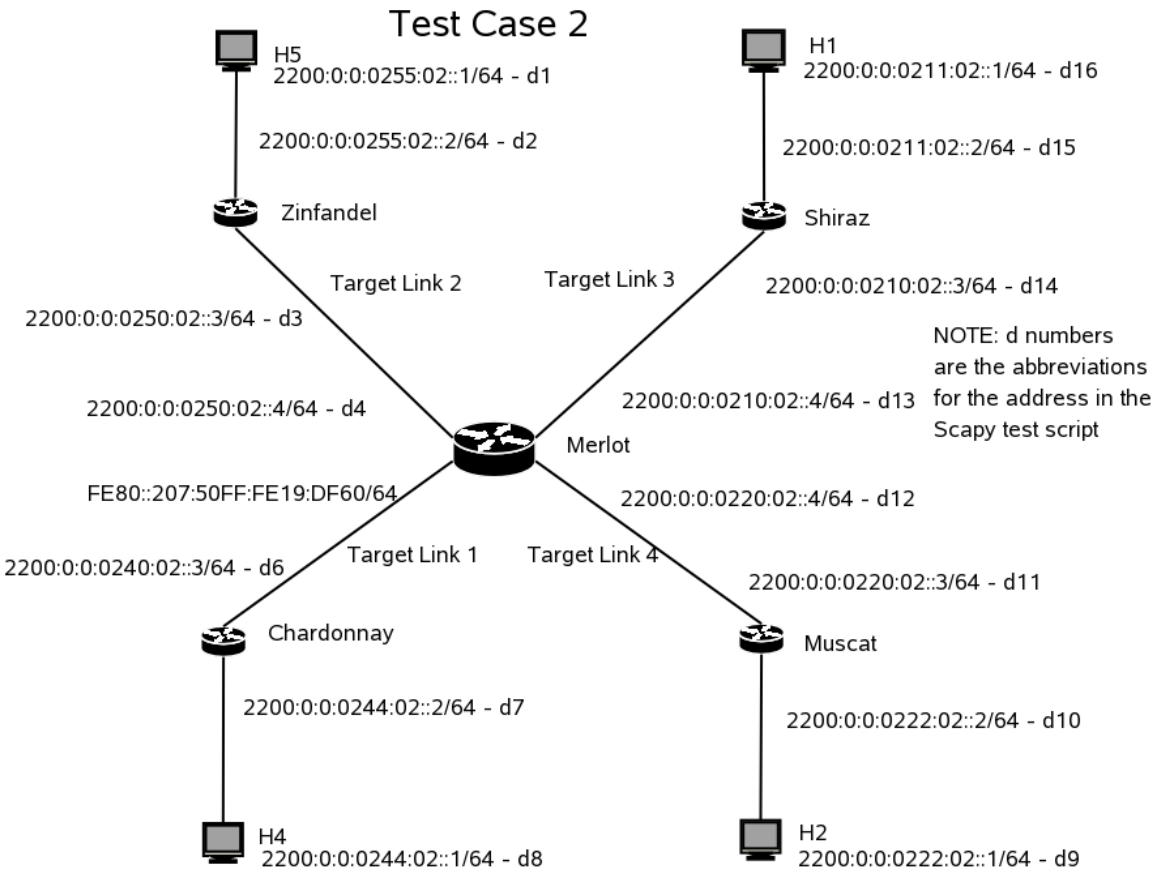

Figure 11 – Lab Test Case 2

2. JUNOS

As with CISCO, we started with UDP. Juniper responded differently than CISCO to UDP probes. A UDP probe to interface d13 from host 4 would always receive an ICMPv6 response from interface d13 provided it had a global unicast address. Juniper seemed to take the definition of a node differently than CISCO in paragraph 2.2 of RFC 4443 (Conta and Deering 2006, 23). Likewise, a probe to interface d5 in Figure 11 would receive a response from interface d5. Time exceeded packets received the same

response as generated by CISCO using the outgoing interface as the response source address. There may be a way to use time exceeded messages for alias resolution but that is left for further study.

Next for ICMPv6 and TCP Juniper responds just like CISCO and the interface pinged uses that address as the source in the response. There were no differences in the responses when compared to the CISCO routers.

However, the unnumbered command works slightly different on JUNOS than on CISCO. Specifically, the command does not exist directly but rather is implied. To use something similar to the CISCO unnumbered command on Juniper, the interface needs to be set up but not assigned an address. Then the interface will use one of two addresses as its default address. First, if the loopback device, lo0, is configured with an address it will always use that address (Garrett, Drenan, and Morris 2002, 912). Second, if the loopback device is not configured it will choose the address of the lowest number interface assigned an address (Garrett, Drenan, and Morris 2002, 912).

To conclude, alias resolution on Juniper routers at this time will not be possible without devising another method either through ICMPv6 quenching or possibly time exceeded. For the purposes of this thesis, this area will no longer be explored due to time constraints and the much smaller number of Juniper routers that we expect to encounter while probing.

3. Why the Difference in Responses?

The differences between Juniper and CISCO are allowed by RFC 2460 and its definition of a node and router. The RFC states that a node is, “a device that implements IPv6”, while a router is, “a node that forwards packets not explicitly addressed to itself” (Deering, Hinden, and Conta 1998, 39). This means in Juniper they can consider the entire router a node so a packet destined to interface d13 in Figure 10 will get a response from interface d13 no matter the outgoing interface and be in compliance with paragraph 2.2(a) of RFC 4443 (Conta and Deering 2006, 23). As for CISCO, they consider interfaces a node and the router a separate node and since the receiving interface

generates the response it can use its own address and this leaves them in compliance with paragraph 2.2(b) of RFC 4443 (Conta and Deering 2006, 23) and, more importantly backward compatible.

D. ANONYMOUS RESOLUTION

1. Responses

One of the items mentioned in the ATLAS paper was that administrators may shut off ICMPv6 to make administration simpler and thus create an anonymous router (Waddington et al. 2003, 59-68). While it is possible to block ICMPv6 with firewall rules, we found no such built in command in the versions of JUNOS or CISCO IOS we were using as implied in the ATLAS paper (Waddington et al. 2003, 59-68). While administrators may block ICMPv6 it still remains to be seen if administrators would bother on backbone routers. It was observed that there is a command to set the point to begin quenching ICMPv6 messages; this was not explored since we do not plan to use this as a method of alias resolution.

It was found during experiments that it is possible to make anonymous routers. If a router is not programmed with any global unicast address of any kind it will not respond with unreachable messages. This was confirmed by programming Merlot with only link local addresses. This means all packets with expired hop counts and packets to ports not open will not receive a response. We were unable to find a way to duplicate the condition mentioned by ATLAS, though, where a router would use the packet's original destination address as the source address for its ICMPv6 packet (Waddington et al. 2003, 59-68).

Finally, as expected, we did not find any packet that would coax a response from a router (Juniper or Cisco) without a global unicast address. Furthermore, the condition mentioned above was the only time, without firewall rules, we found that you could truly create an anonymous router.

2. Bisimilarity

For the purposes of this thesis, we will use a method close to bisimilarity mentioned in Yao's paper to conduct anonymous resolution. The method is described as, "two anonymous nodes that are adjacent to the same set of known routers can be identified as being the same (Yao et al. 2003, 353-363). While this method is not as good as the methods described later in his paper, implementation of his better methods will be left as future work. For example, in Figure 7 we could assign known node 2 to X and known node 5 to Y in a threesome X-A-Y with A being anonymous. Then we could match this threesome against all threesomes where X and Y match and the A node is anonymous and replace all matches with a single threesome as in Figure 12 and call the anonymous node 4. Using this method of bisimilarity we would reduce the all the anonymous nodes in Figure 7 to the map in Figure 12.

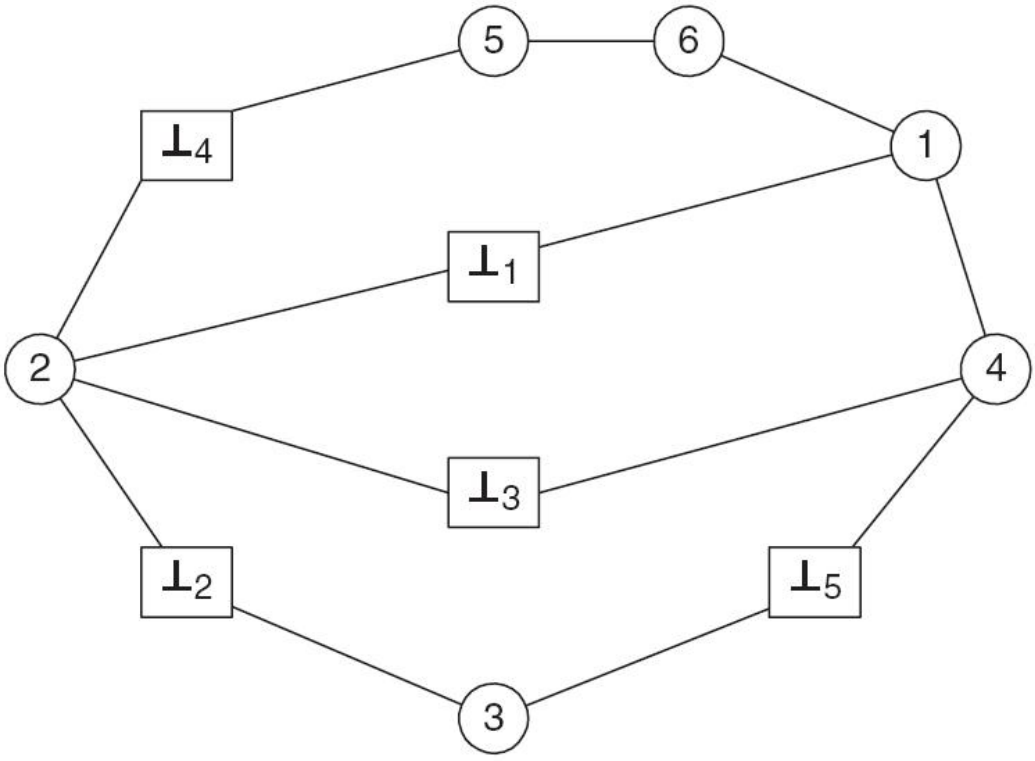

Figure 12 – Bisimilarity (From: (Yao et al. 2003, 353-363).)

E. SOURCE ROUTING

1. Capability Testing

Testing for source routing capability is very similar to the method mentioned in the Mercator paper, but in IPv6, responses are slightly different (Govindan and Tangmunarunkit 2000, 1371-1380). Consider the network in Figure 12, if we send a source route probe to d15 with its final destination as interface d13 we will get one of two responses. If source routing is enabled on Shiraz then we will get a port unreachable response from the appropriate interface on Merlot. On CISCO routers if source routing is disabled through the use of the "no source route command," we get an unreachable message from d13 with the code set to "administratively prohibited." JUNOS contains the same command for disabling source route, but as of release 8.2R1.7, JUNOS does not stop IPv6 source-routed packets. Setting the no-source-route option on JUNOS stops IPv4 source-routed packets but has no effect on IPv6 packets.

2. Disabling Source Route

In order to facilitate MIPv6 administrators are not supposed to completely block routing headers used for source routing (Convery and Miller 2006, 43). This is so that a routing header can make it to the home agent and then the packet can be rerouted to the actual destination. While this may be needed, unfortunately, in most CISCO IOSes up to and including the first releases of 12.4, there is a vulnerability that allows remote users to crash a router with a specially crafted packet. Due to this vulnerability, CISCO has advised their clients to firewall routing headers in IPv6 (Anonymous2007a). It is hoped that this advisory has not shut off all source-routing capable routers.

F. ADDRESS SPACE PROBLEM

One of the many problems when mapping IPv6 is the number of potential addresses, as explained in the previous chapter. One of the goals of this research is to try to reduce the problem to a more manageable level.

Fortunately, during research we realized that the address space problem was not as big of a problem for router level topology as once thought. The rationale for this goes as follows. First, the last 64 bits of the address are used to identify a host on a subnet and thus are not part of the equation when mapping router level topology. Provided we probe every subnet allocated to an AS, we will reach every router on the subnet. This means that for the most part, ASes will be typically assigned a /48 or less subnet, tending to the "less" side. So, for most cases, there will be (128 – 64 – 48) bits prefixes, or 2^{16} subnets to probe. While this is a lot for the large cases, it is still only 65536 subnets and doable for most probing engines. The only area that is not addressed is locating the interfaces servicing the subnets. This area remains a problem because it was found that for both CISCO and JUNOS their default behavior is to not return an unreachable message when you probe for an address that is not in their serviced subnet. For example, a probe to 2200:0:0:0211:02::55 would not be answered by Shiraz in Figure 10. Methods to harvest these addresses from Google and other sources are also left as an area of further research. Finally, while not an issue with this thesis, any attempt to locate hosts on a subnet still has to deal with the problem of probing 2^{64} addresses and this is no small feat.

In conclusion, since there was no need to locate a method of parsing addresses down to a manageable level we felt that the probing engine would work effectively through self-discovery. Self-discovery means that when one source locates a new node all the other sources will also probe this new node to locate new routes to that node and add the new nodes they discover to their lists of nodes to probe.

G. ADDITIONAL NOTES

Noted during testing is an error in the Wireshark protocol analyzer formally known as Ethereal (Combs 2007). The error concerns IPv6 packets containing a routing header with remaining segments. As per RFC 2460 paragraph 8.1, when calculating checksums "(i)f the IPv6 packet contains a Routing header, the Destination Address used in the pseudo-header is that of the final destination(Deering, Hinden, and Conta 1998, 39)." The rationale behind this is based on the fact that routers do not calculate checksums so the incorrect checksum during the period the packet traverses the network

will not be noticed but when the packet arrives at the destination the checksum will be calculated based on the address in the destination field of the packet at the time it arrives, which is the address in the last segment. As of this writing, Wireshark uses the address in the current IPv6 packet without regard to the number of segments remaining and thus flags outgoing packets as having an incorrect checksum. This error has been reported to Wireshark's bug tracker for correction in the future.

IV. DESIGN AND IMPLEMENTATION OF PROBING SYSTEM

A. INTRODUCTION

The goal of the thesis was to develop an engine that would automatically probe a target AS in an IPv6 network and develop a list of edges using the minimum number of probes while still permitting the discovery of nearly all the same nodes and links found by traditional methods (Donnet et al. 2005, 327-338). Due to the complexity of the system, it was decided to implement the system in three stages initially and then, possibly for future work, to pull all the stages together as a single engine. The first stage is the probe engine, second is the alias resolver and, finally, the anonymous resolver stage, as depicted in Figure 13.

The main probing engine's purpose is to obtain the list of edges while reducing the probing load through the use of the Doubletree algorithm. An edge is defined for the purpose of this thesis as two directly connected nodes, each with a unique IPv6 address. Either or both of the nodes may be anonymous, the system will ensure each anonymous node has a unique IPv6 address and this methodology is explained later. The edge list is the list of all the edges the system has found and when put together compromises the connected graph forming the network in question. The probing engine has numerous configurable parameters on startup to include timeout for anonymous nodes, number of probes for each destination, number of responses needed to consider a response valid and cumulative distribution function (cdf) percentage required for deciding the initial hop count (i.e., TTL) to use for the probes.

The alias resolver loads the edge database created by the main probing engine and sends UDP probes to all the individual nodes from all the probing sources in an attempt to resolve as many of the nodes to a small set of routers as possible. Once done, the resolver saves the original file to a backup and then writes a new edge file with duplicates removed.

Finally, the anonymous resolver is executed completely offline. It takes the edge file output from the alias resolver, eliminates duplicate anonymous nodes through the use of bi-similarity, and creates a third and final edge file. The next few sections describe the three stages and their major parts and functions that together make the entire system.

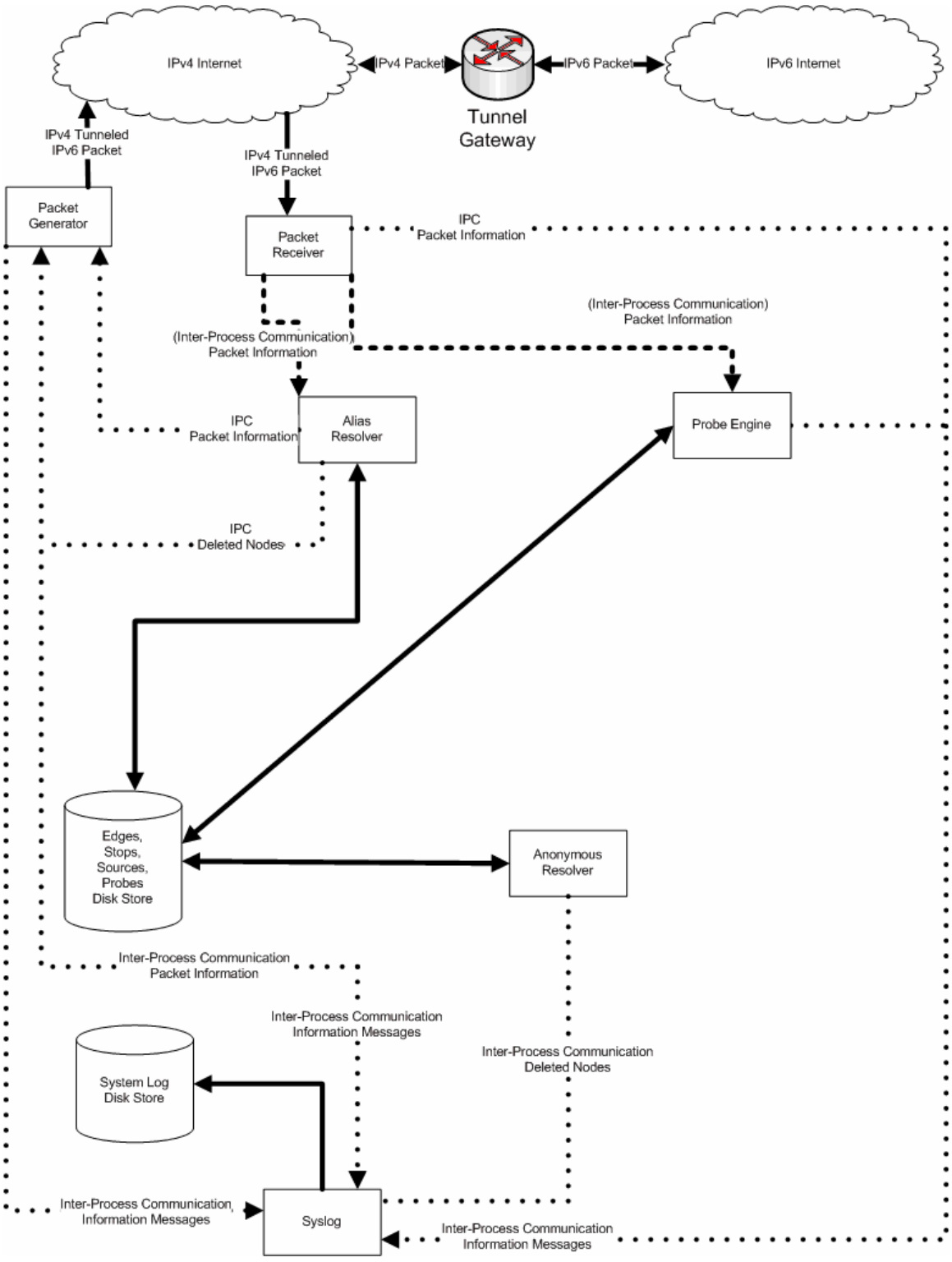

Figure 13 – Probing System Overview

B. PROBING ENGINE

Before describing the engine a little background on key terms and their relationship to the methodology is required. All descriptions refer to **Error! Reference source not found.** and are used to describe a typical probe using this system. When the engine first starts, the initial hop count must be determined to eliminate it from the calculations. The initial hop count is the number of hops from the tunnel broker, the point where the IPv6 packet is de-encapsulated and enters the IPv6 network, to the source router. NOTE: For the remainder of this thesis the source router will be referred to as the source and each source router resides inside a different AS surrounding the target AS.

After the initial setup is complete, probes all begin at the point *h* in Figure 14. The value of *h* is calculated by taking the command line specified percentage of the cdf on all the hop-counts of nodes previously discovered by this source. For example, suppose the percentage specified on the command line was .05 or 5% and the source had received responses corresponding to the hop count, i.e. 1 for 1, 2 for 2 and 3 for 3 for all hop counts 1 through 10. Taking .05 of the 55 responses received (sum of all responses 1 through 10) you would get the cdf value of 2.75, the engine then would start through the array adding each low value until it exceeded the cdf value. For this case it would add the hop count 1 and then 2 giving a total of 3, this would cause it to select the hop count of 2 as the *h* value. So for a complete example using Figure 14, let's say the initial hop count was 3, in other words there are 3 hops from the tunnel broker to the source, this would mean all probes would begin at the hop count 5 or the point marked *h* in Figure 14.

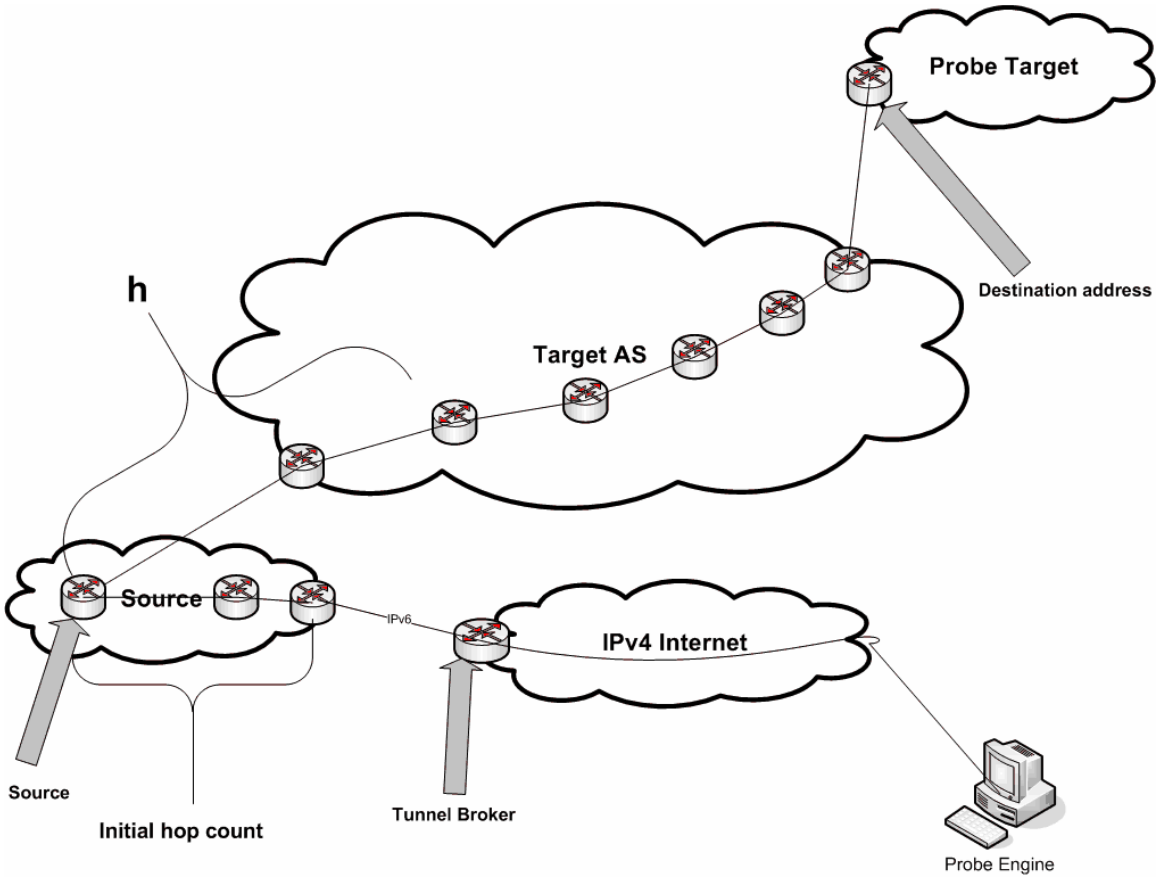

Figure 14 – Typical Probe

Initially all probes would begin at *h* and the engine will increment the hop count moving towards the destination until it reaches one of three conditions: it reaches the destination; or it reaches the maximum number of anonymous nodes allowed which was specified on the command line at start up; or it reaches a previously recorded edge, or a global stop, for this destination. Once one of these three conditions is met, the probe engine calls *change direction*. The *change direction* function causes the engine to start at *h* again and probe backwards towards the source. After the change of direction, the engine will decrement the hop count until one of the following three conditions is met: it reaches the source; it reaches the maximum number of anonymous nodes allowed; or it sees a node previously recorded for this source, i.e., a local stop.

There are two special cases that have not been covered until now but are very important for the probing engine. The first case is where the router at *h* is non-responsive

that is it is anonymous. In this case, it always moves the hop count towards the source and restarts the probes. This reduces the complexity of the engine. The engine repeats this scenario until the initial node for probes is not anonymous or until it reaches the actual source. If the probe engine reaches the source, it then marks the probe as source-started and begins probing towards the destination. When it reached one of the conditions for terminating the probes moving towards the destination it will call the *finish with destination* function, which is explained later.

The second case is when the response to a probe is any type of unreachable message. In this case, the destination is marked "bad" for this particular source and during later processing the destination is removed from the probe list for this. The probe list is a list maintained by each source containing future destinations to probe and is populated by other sources. When a source finds a new node, it adds this node every other source's probe list for later route discovery.

As stated before the engine's main purpose is to reduce the redundancy of probes while obtaining nearly the same set of nodes that would have been discovered using more traditional probing techniques (Donnet et al. 2005, 327-338). It accomplishes this through the use of Doubletree's methodology of starting in the middle of a path (Donnet et al. 2005, 327-338). Figure 15 provides insight into the structure of the engine. Each of the primary components of the engine, as depicted in Figure 15, is discussed below.

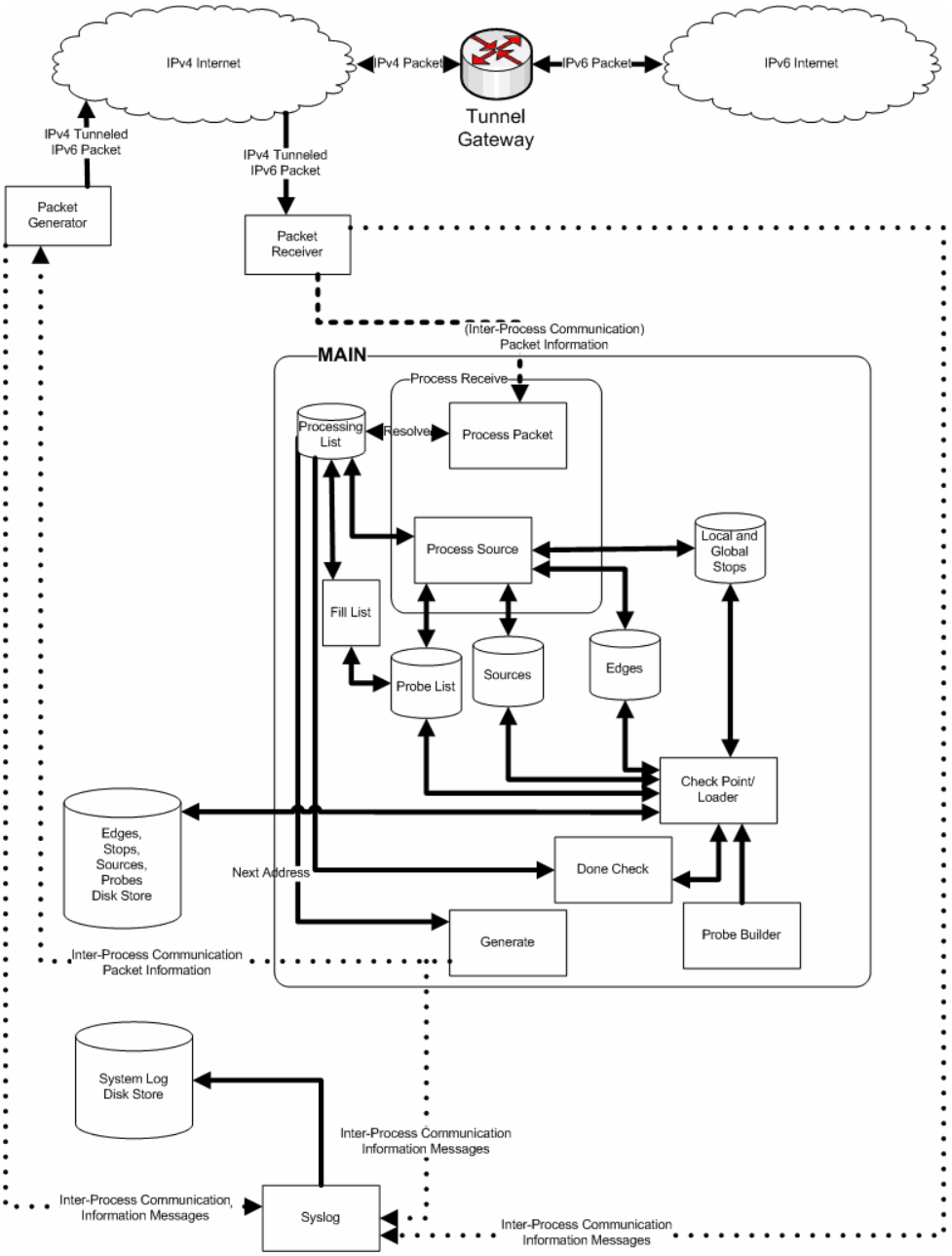

Figure 15 – Probing Engine

1. Main

This module encompasses many sub-modules and also contains the startup functionality of the system. It first allocates all the data stores in memory to include the edge list, stop list, source list (which contains pointers to the probe list) and finally the processing list. The edge list is a data store containing all edges discovered by the

probing engine, in the format of the two nodes that comprise the edge, the source that discovered the edge, the destination being probed when the edge was discovered and the hop count from the source to the last node of the edge. The stop list contains, a global-stop-set telling the engine when to stop sending "outgoing" probes and a local-stop-set telling the engine when to stop sending "incoming" probes. The source list, all probe sources, their initial hop counts from the tunnel entry point, their IPv6 address and an array with entries that correspond to the hop-count of all the replies for use in calculating *h*. The source list also contains pointers to the appropriate local stop set and the probe list for that source. The processing list contains one entry for each source containing the destination information needed to process returning packets. This list is passed to many other routines so they can make proper decisions. It is also used to maintain state information.

One of the keys for the Doubletree algorithm is the value of *h*, however, on initialization the engine has no values for hop-counts since it has not yet discovered any nodes. To overcome this, after the engine discovers the initial hop-count for a source it and then probes all the destinations in the initial probe list, as built by the probe builder module discussed later, to determine their hop-count from this source. The only knowledge gained during this probing is the hop-counts for calculation of the value of *h*. Once all the destinations in the probe list are probed the engine determines the initial value of *h* and stores it in the processing list and then moves to the next source. The engine repeats this process for all sources until each has an initial value of *h*.

2. Process Receive

This, by far, is the most complicated module in the entire probing engine. It contains two sub-modules called process packet and process source.

a. Process Packet

This is the simpler of the two sub-modules in process receive. This routine takes the inter-process messages from the packet receiver and determines to which source it belongs by comparing the ICMPv6 sequence and ID number of the

received packet to those stored on the source list by the generate function, which is explained later in this section. Upon finding a match it increments the response count of that source.

There are four special cases for this function. First, if the packet returned from a source is "route unreachable" or some other "unreachable" message, then this routine sets a "bad" flag in the process list for the appropriate source to be handled by the process source module. Second, when the routine receives a response from two different nodes for the same set of packets, suggesting route instability, it records the two nodes and the final destination in the log but does not increment the response count. Third, when the routine receives a packet from a destination not being probed, possibly a delayed response from a prior probe, the module just notes the packet in the log for future analysis. Last, the routine may receive a packet indicating that it was returned with remaining segments suggesting the hop-count expired before reaching the source, possibly due to route instability; in which case, the system just records the event in the log.

b. Process Source

For each list of responses built by process packet, each keyed to a particular source, process source develops an edge for insertion into the edge list. The set of edges are the basis of the ultimate product of the engine, a map of the autonomous system probed. The processing, as described below and illustrated in Figure 16 through Figure 20, is repeated for each source, incrementally developing probe sets for each source until sufficient data is gathered to extrapolate the edge list.

Figure 16 addresses two cases. CASE A handles two special instances. First, if the "bad" flag is set for the source, the module will call *finish with destination*, explained later. The second instance is when the engine has probed beyond a maximum count, when probing toward the destination, or probed beyond the expected distance to the source, the "minimum" distance, when probing away from the destination.

If the probe engine has gone beyond the maximum number of hops allowed it will check the source start flag, which is used to identify probes started from

the source and not from *h*. If the source start flag is not set it will change direction, and probe back towards the source being evaluated by starting at *h* and incrementally decrementing the probe TTL, otherwise it will call the routine *finish with destination*. However, if the engine has gone beyond the source when probing towards the source then it will call *finish with destination*.

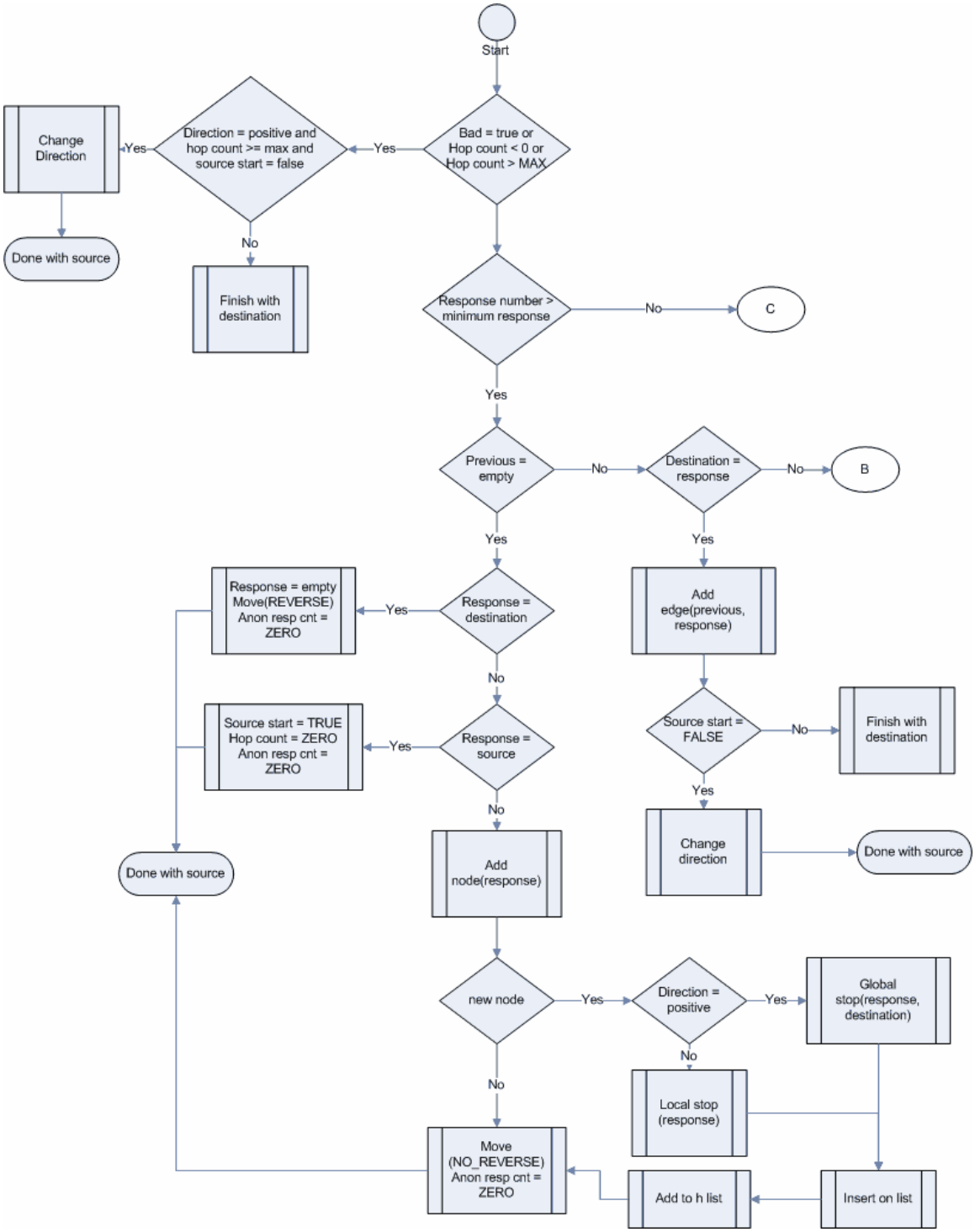

Figure 16 – Case A and Beginning of Case B

CASE B addresses those responses corresponding to the most typical outcome of a probe. It starts by checking if this probe is the first in its direction, since

Doubletree starts probes in the middle and not at the source. If so, and it reached the destination, the probe TTL must be reduced and the probes continued such that the topology between the source and the destination can be interrogated. The *move* function accomplished this TTL adjustment, using two settings: if NO_REVERSE is set it will just move the hop count in the appropriate direction towards the destination for outgoing or towards the source for incoming; if REVERSE is set, the function causes the probing to go backward, toward the source, by reducing the TTL value. The other functions of *move* are to set the response's source as the end point, or vertex, of a new edge and prepare the trigger for the next probe.

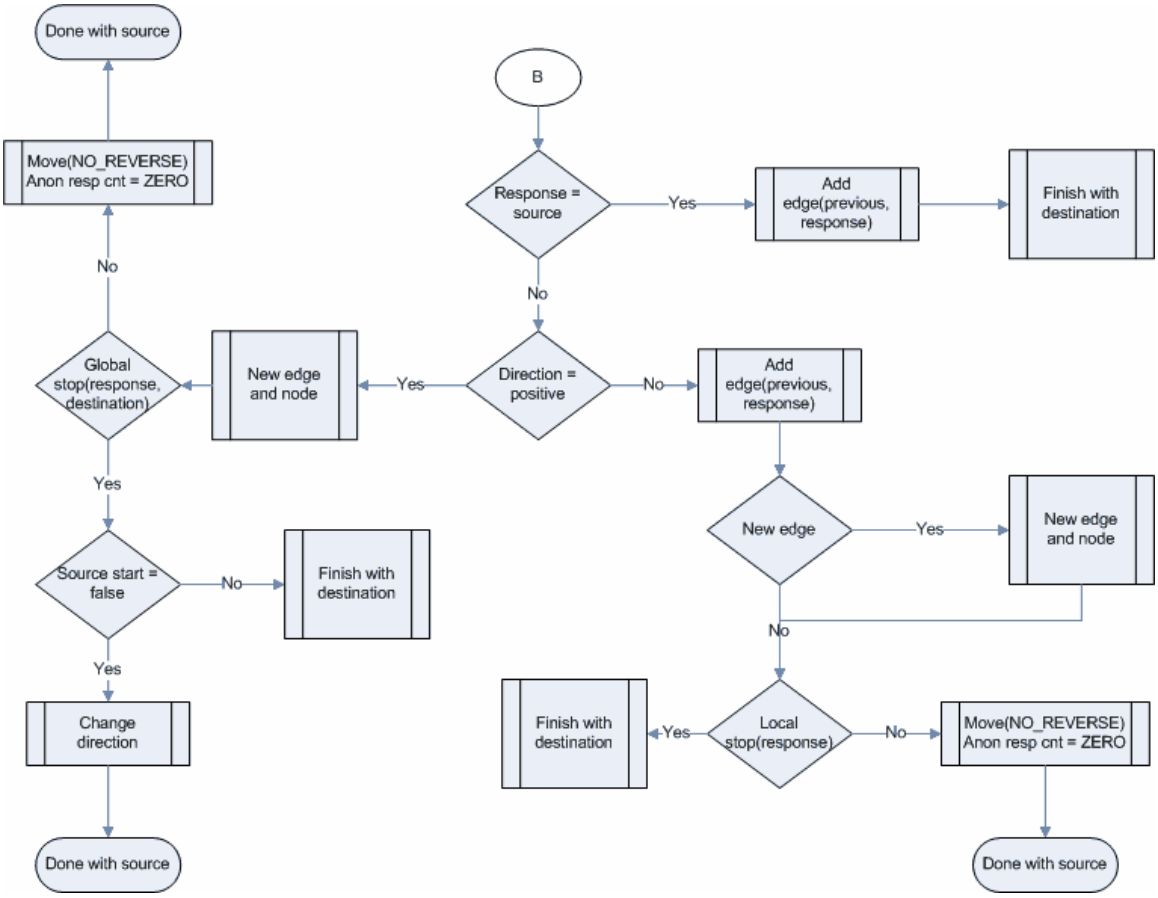

Figure 17 – Case B

If the first probe hit does not reach beyond the source, then it is possible the value of *h* for this source is zero, due to a limited number of responses, or the destination is only one hop away from the source (recall that a destination may be a node

discovered by another probe source and is inserted in every other probe source's probe list). At this point the engine identifies the source as the first vertex of a new edge and continues probing the network, increasing the probe distance with each probe, to determine the other vertex. However, since the probes have already interrogated the topology close to the source there is no need to work backward from the value of *h*, thus the probing ceases for that particular source when the probes reach the destination. To track this situation process source sets the "source-start" flag.

If an edge has been initiated, by identifying one vertex, but is not complete, process source checks to see if the engine has hit the destination. If so, the destination is set as the distant vertex and the edge is added to the edge list. If the topology closest to the source has not been probed, probing restarts at *h* and continues according to the Doubletree methodology, until the TTL reaches zero or a local stop set element is reached.

Most typically, the probe does not result in the identification of the start or finish of a series of edges making up a path from the source to the destination. In this case, the new edge is added to the edge list and the new vertex is set as the origination for the next edge. To reduce the probes necessary, process source then checks to see if the node has already been placed in the appropriate stop set, according to whether the probes are interrogating away from the probe source or toward it. If it finds the node in a stop set the topology has been probed beyond that point and probing in that direction should stop. If probing towards the source and a local stop set pair is found that includes the node then all probing for this destination from this source is complete. If a matching stop pair is not found then one is created for the node and inserted into the appropriate list. If no stop set pair is found then process source checks to see if the interrogated node is new. If so, process source calls *Insert on list* to add the node to every other source's probe list. This is how the probe engine performs self discovery. Since it is a new node it adds the nodes hop count to the h list which will be used later to calculate a new value for *h*. Finally, process source calls the function *move* to modify the probe hop distance such that the next node may be probed.

Case C is the special case that deals with anonymous. If during the pass, the wait time for the node has been exceeded without the minimum number of responses being received then this case executes. If no new edge has been and if the hop-count is zero, process source sets the source start flag and initiates a new edge with the source as the vertex and calls the *move* function. If the hop-count is not zero, it calls the *move* function with the REVERSE flag set telling it to probe closer to the source.

The normal instance for CASE C is when an edge has been initiated but is yet to be completed. If the limit on the number of anonymous nodes included in an edge is reached it calls *finish with destination.* Otherwise, an edge is created with the current anonymous node number and the function increments the system anonymous node number. After this, it adds the edge to the edge file and makes the decision on the next course of action. If the number of anonymous responses has not exceeded the threshold and the hop count is within limits, it calls the *move* function. Else, it checks whether or not the nodes closest to the source have been interrogated. If not, it calls *change direction,* otherwise it calls *finish with destination.* *Change direction* restarts the probing at h and probes towards the source.

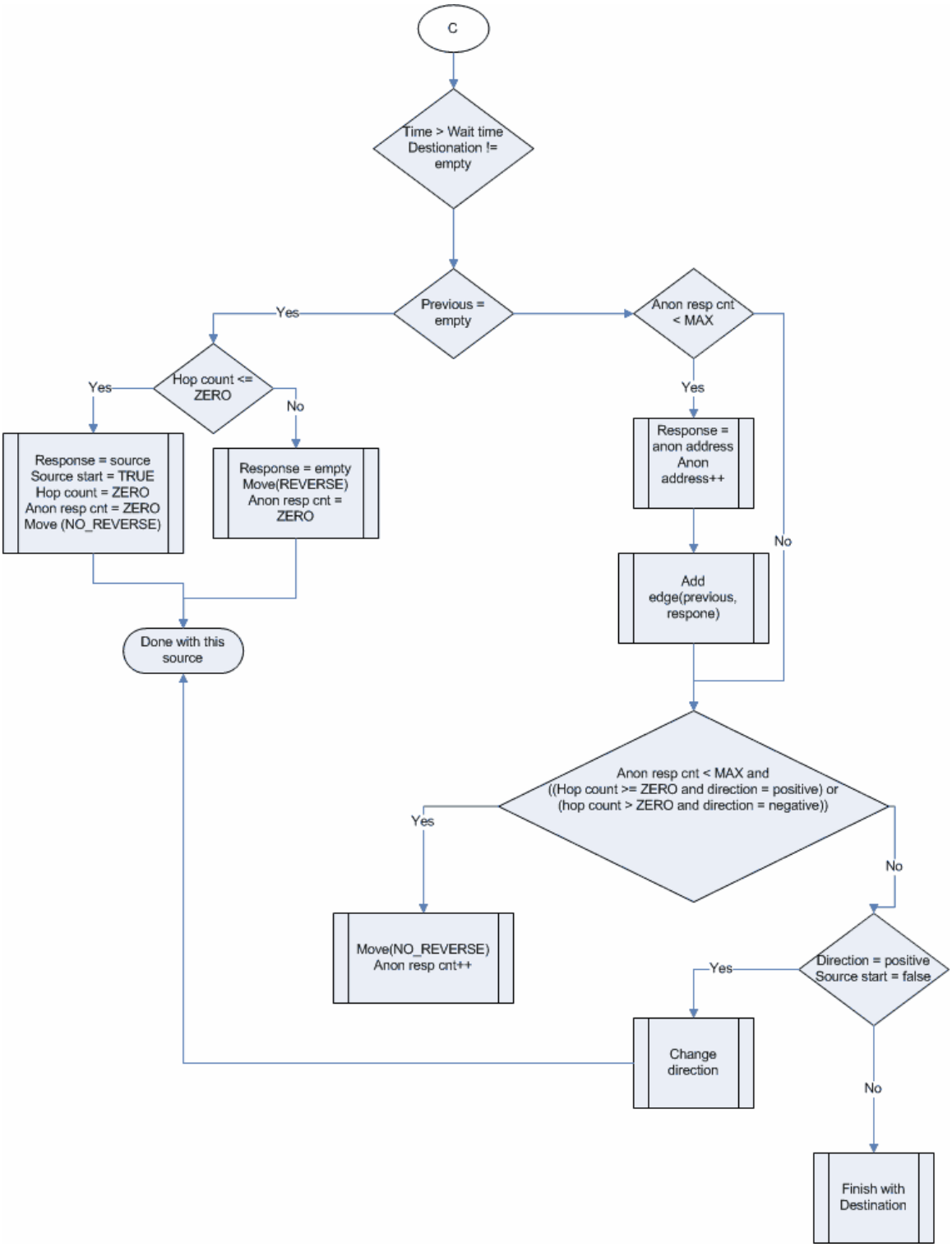

Figure 18 – Case C

There are two special functions contained throughout the diagrams depicting the process source module, they are explained here. The first is *new edge and node*, shown explicitly in Figure 19. This function first checks to see if the edge is new because this is a lower cost operation than checking for a new node. If the edge is new it checks to see if the node is new, and if it is it calls *insert on list*. The second function is *finish with destination*, as shown in Figure 20. This routine first eliminates the current response node from the probe list and the processing list for the source. Then it recalculates the value of *h* for the source. This calculation is only done here because if it were done in the middle of probing a destination it could cause the hop count to be wrong when the direction was changed thus missing an edge.

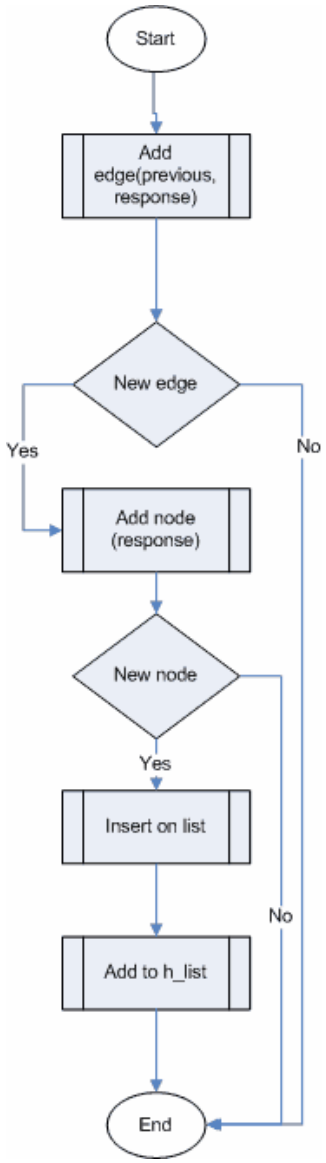

Figure 19 – New Edge or Node

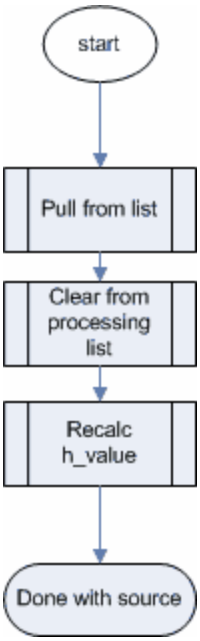

Figure 20 – Finish with destination

3. Checkpoint and Loader

This module is actually two functionalities but due to their similarities their explanations are combined into one. The purpose of the checkpoint and loader modules is to ensure the ability to recover from a crash of the probing engine. The probing engine runs with all of its data files residing in memory for speed efficiency. Unfortunately, if the system crashes, for whatever reason, all of these files are lost. To prevent the loss of many hours of probing, the system has the ability to dump all of these data files to disk in a text form for later editing, revision and restarting.

When the system is started there is a parameter that tells the system how many minutes of probing to conduct before checkpointing all the files. The system then invokes the loader, which reads the disk files and loads them into memory. After the required number of minutes have elapsed the system calls *checkpoint*, which renames the previously loaded files appending the current time and day of week to their filenames and changing their extension to .bak. The checkpoint function then writes all the files to disk with the latest information and closes them. This means if the system crashes before the next checkpoint it can be restarted and the loader will load the last set of checkpoint files.

There is one exception. Since the processing list is very sensitive to the time of capture it is not sent to disk and it is not part of any of the disk files. This means the stop file, which is accurate to the second, could cause a node to be missed upon reload due to differences in the stop file and the edge file, which defines path. To prevent this problem from occurring, the stop file is not reloaded upon restart. This method ensures the most accurate map of network and has minimal impact as the stop file will quickly rebuild. NOTE: To support testing there is a flag that can be set in the loader to disable this behavior, thus allowing the last stop file to be loaded.

4. Fill List

This utility puts new destinations on the process list for a source when that source has an empty process list. It cycles through the entire process list for all the sources and when it finds an empty destination it extracts a random destination from the probe list for that source and puts it in the process list as the destination for that source. It then determines the hop count for that destination by taking the initial hop count for the source and adding it to the value of *h* for the starting hop count and sets the generate flag to generate a new probe sequence. NOTE: if the value of *h* is zero, it then establishes a new edge with the source as the first vertex and sets the source-start flag. Finally, if the probe list for a source is empty then fill list will leave the destination empty for that source and not set the generate flag.

5. Done Check

This routine runs immediately after the *fill list* routine and checks to see if every source has an empty process list. If so, then all the destinations have been processed from the probe lists and scanning is done, once this condition is reached *done check* ends the program.

6. Generate

This process generates new packets when triggered by another module. It cycles through all the sources on the process list and, if the generate flag is set for a given source

it will generate the proper inter-process communication message to the packet generator to generate the required packet. For each ICMPv6 packet, it generates a random number for the sequence and identification fields and stores it in the process list for that source for later identification of received packets. For UDP packets, it generates random source and destination ports for identification and stores them on the process list entry for that source for later identification. It also records the send time on the process list which is used later to determine if the packet timed-out.

7. Packet Generator

This facility generates all requested probes, either UDP or ICMPv6. It can generate source-routed or non-source-routed probes. The number of packets generated per probe is based on the number passed in on the command line at startup. Also passed in on the command line are the tunnel broker's IPv4 address and the engine's IPv6 address. The generator will automatically determine its own IPv4 address for the packets. Every time the generator is started, it also obtains a new random number and uses it as the run number. After startup, the utility takes an incoming message from the inter-process communication system that contains the type of packet, source address, destination address, hop count, port, or id number and sequence number and generates the IPv6 packet, encapsulating it in IPv4 tunnel packet, and then sends it to the tunnel broker's entry point. Using the previously mentioned run number, and managed by a configurable threshold, the generator logs the number of probes sent per that threshold and also records to the system log when shutting down.

Code copied from tcpdump is used to correctly calculate the checksum for outgoing UDP and ICMPv6 packets (Shaw 2006). The module also uses libnet to create the final IPv4 encapsulation packet containing the IPv6 packet (Schiffman 2007).

8. Packet Receiver

This utility listens for probe response packets and then forwards them to *process receive* using inter-process communications. Processes, interacting through inter-process communications, were used to ensure the receive queue does not fill up before a packet

can be processed by the receiver. The *Packet Receiver* parses packets, extracting the type of response, ports, sequence, id, and other original sending packet information, upon which it creates an inter-process message and sends it to the main process. Any packet that cannot be parsed correctly is recorded in the log with its identifying information. The receiver also uses libpcap to receive the messages and filter traffic destined to the engine (Shaw 2006). It also uses code originally contained in tcpdump (Shaw 2006), but modified for a dissector routine the author found posted on the Internet, to dissect the returning IPv4 packet (Casado 2001).

9. Probe Builder

The probing system requires four initial files in order to properly execute the scan. The files are probe_list.txt, probe_source.txt, probe_stop.txt and probe_edges.txt. Each is an ASCII text file that can be created by hand using any text editor. However, their format is based on the structures in the program, so to prevent problems when creating the files a utility called *probe builder* was created to interactively create these files. As the probe engine performs self-discovery by making every source the destination of all other sources, destinations are not needed but they will increase the resolution of the resultant map. The program accepts input from stdin, so the input for probe builder can be redirected from a file, provided the proper sequence is followed.

10. Syslog

The purpose of this module is to record to disk significant events of the probing engine. This module runs as a separate process to ensure maximum efficiency of the system. The system uses inter-process communications to communicate with the syslog utility, which records everything to a file named probe_syslog<date>.txt. The logging facility will close the file and open a new file when the date changes at midnight. However, if the main system crashes the system logging facility will stop recording at that point as the main process owns the message queue. Thus, anything in cache at the time of the crash will not be written to disk.

To compensate for this while troubleshooting, there is a flag that can be set in the syslog source code that will force a flush to disk for every call ensuring every message is in the log; but this feature does slow the entire system down. Finally, writing a custom system logging facility supports portability of the engine.

C. ALIAS RESOLVER

The purpose of the *alias resolver* is to reduce the number of aliases for a router. The resolver sends UDP probes from all the sources in an attempt to obtain all the addresses a router uses for its interfaces. As mentioned in previous chapters, this will only work on CISCO routers and has no effect on Juniper. A number of routines, originally developed for the probe engine are also used, so only differences in the resolver will be mentioned here. All references to parts of the resolver refer to Figure 21.

One major difference in the alias resolver, as compared to the probe engine, is the wait function. In *alias resolver* the wait is for all sources and is not based on responses. The probe engine generates packets as fast as it can process and as fast as the destinations can respond. The alias resolver, however, sends packets and waits a prescribed time before moving to the next batch. This was done to simplify the code and processing.

1. Checkpoint and Loader

The major difference here is in the loader. The loader extracts the nodes from the edge lists, inserting non-duplicates and non-anonymous nodes into a node list.

2. Fill List

For *alias resolver, fill list* goes through the node list and locates the next node that has not been resolved and loads that node in the processing list for every source. It also sets the resolved address for that node to the original address. Finally, it sets the generate flag so the generate function will generate a UDP probe.

3. Process Alias Packet

This function is much simpler than its counter part in the probe engine. This function processes all the inter-process messages from *probe receiver*. It matches the responses to the source based on the source and destination port contained in the unreachable message returned. The routine then increments the response-count for the source and records the source of the responding ICMPv6 port unreachable message.

4. Process Alias Source

This routine is extremely simple compared to its counter-part in the probe engine. For any source that has received the minimum number of responses, it takes the responding address and sets its resolved address to the destination address. When finished with a node, it marks the source in the process list as requiring another destination so *fill list* will get another node for the source.

5. Generate

The only difference between this version and the probe engine version is this version generates requests for UDP probes and assigns random ports above 1024 to the source and destination ports.

6. Resolve

This is the heart of the routine. After all the nodes are resolved, the routine then loads each edge from the database and for each non-anonymous node it replaces the node's address with the resolved address in the node database and attempts to store the edge using the same routine the probe engine uses to store edges. The routine to store edges rejects edges that are duplicates and informs the calling routine so it can record this information. Once all edges are saved in memory the routine calls the checkpoint routine to backup the old file and save the new edge file to disk.

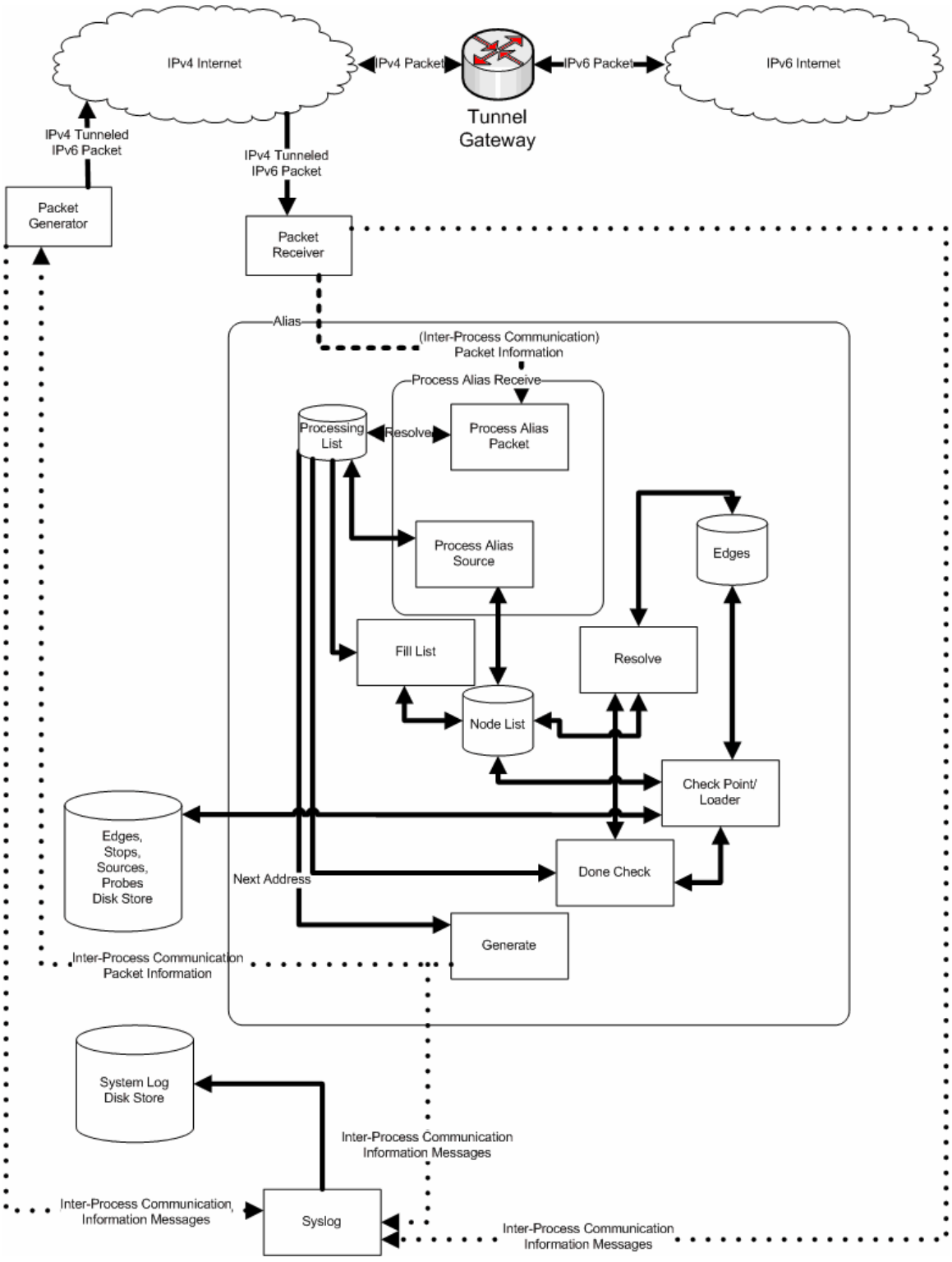

Figure 21 – Alias Resolver

D. ANONYMOUS RESOLVER

The anonymous resolver is much different than all the other parts of the system because it runs completely off-line. It does use several of the same functions from the probe engine so only differences or points that require emphasis will be mentioned here.

As mentioned earlier in the thesis, only bisimilarity will be used for anonymous resolution. All references to functions in the description below refer to Figure 22.

1. Checkpoint and Loader

This uses the same routine as the probe engine to load all the edges and also uses the same routine to checkpoint the edges. The changes happen externally to the routine, so its functionality could be used without change.

2. Resolver

This is the heart of the routine. This routine immediately renames the edge list in memory and creates a new edge list that is empty. It then searches the old edge list for any anonymous nodes. When it locates one it locates its X and Y nodes as explained in the bisimilarity section of the thesis. Upon locating the X and Y nodes it searches the entire edge list for a threesome with the same X and Y and an anonymous node in the middle. If it locates a threesome, it replaces the anonymous node in that threesome with the anonymous node from the first threesome. It repeats this routine for all the anonymous nodes. It then sequentially adds all edges to the new edge list using the same routine from probe engine that eliminates duplicates. Upon completion it calls checkpoint to backup the old edge file and create a new edge file with the bi-similar threesomes removed.

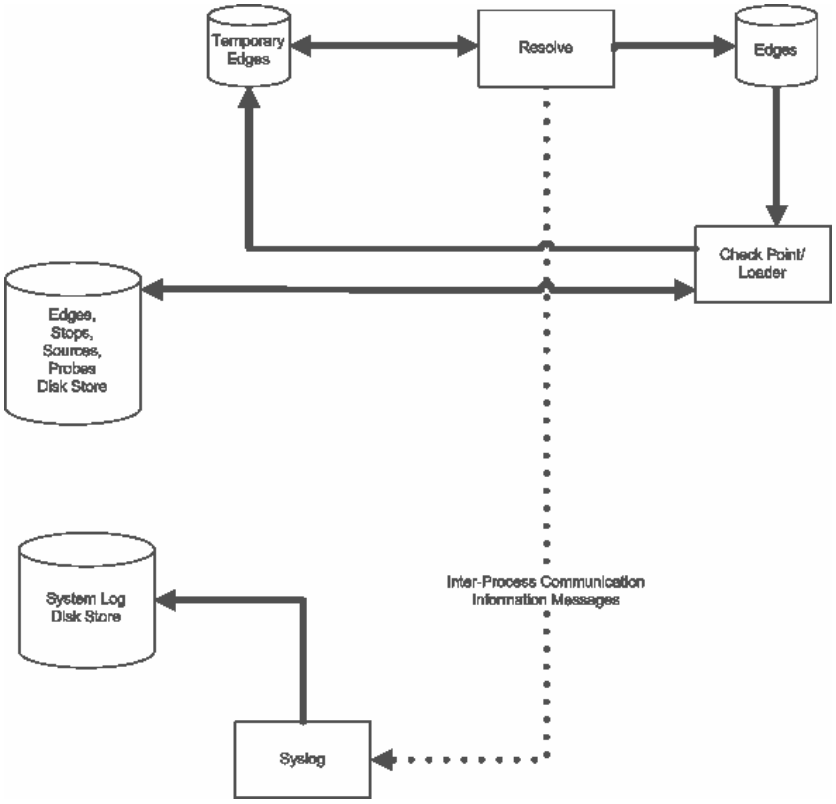

Figure 22 – Anonymous Resolver

E. CREATION OF INITIAL PROBES

For the probe engine to work, an initial set of probes that have a high probability of traversing the target AS is required. A set of routines were developed to create the initial set of probes used by the probing engine given a particular target AS number. All of the initial probing information (mainly the source and destination IPv6 addresses of the probes) was created using data from the RouteViews project (Meyer 2007). All of the tools can be used with data from other sources, like RIPE Network Coordination Centra (Anonymous2007b). Creating the files needed for the program to run was done using the programs below in the order represented. Note everything explained below is based on a Linux operating system, users of other operating systems will have to adapt these instructions.

1. Zebra-dump-parser

This is a freeware program available from the Italian Linux Society (d'Itri 2007) that converts the binary Multi-Threaded Routing Toolkit (MRT) format data files from the RouteViews website to readable ASCII text (Meyer 2007). Once downloaded, the Zebra dump parser is simple to run but requires Perl. In order to obtain IPv6 data, the file zebra-dump-parser.pl must be opened and the settings $ignore_v6_routes must be set equal to zero and $format set equal to two. To use the program, type "bzcat the-rib-filename.bz | time | zebra-dump-parser.pl > DUMP 2> DUMPERR. Note that the names DUMP and DUMPERR are the user's choice and can be changed, but those names are used as references later in this thesis.

2. My_parse

This utility is a small C utility created by the thesis author that parses the DUMP file into files useful for processing. For example, to parse AS 2999 type my_parse DUMP 2999. This will parse the DUMP file, eliminating all the IPv4 routes and all those not containing the target AS. It will also eliminate all routes to other ASes that originated from the target AS. The program creates three files, peerfile.txt, asfile.txt, and allfile.txt, which are detailed below.

a. Peerfile.txt

This file contains the destination prefix that took the packet through the AS. It contains three more columns. The first is the egress AS when following this route. The second is the target AS. The third is the ingress AS for getting to this AS. In other words, if you were to use a router in the entering AS as the source and an address from the destination prefix as the destination it would traverse the target AS and the egress AS on its way to the destination prefix.

b. Allfile.txt

This file contains a list of all ASes that surround the target AS. Note, this list is not sorted or checked for duplicates, which will be fixed in the next step. This file is the most useful for the engine.

c. Asfile.txt

Since the target AS is the source of some prefixes, this file contains those prefixes that belong to the target AS. This file is useful for generating addresses that will increase the lower level resolution of the probing engine.

3. Sort

The parse utility does not sort or eliminate duplicates. There is a Linux utility to accomplish this task. To accomplish this next step take each text file created by the parse utility and execute **sort -g -u thefilename.txt > newfilename.txt**. This will create the files needed for the next step.

4. Traceroute6

Using the sorted allfile.txt, we sent probes using traceroute6 engines available through a simple search on the Internet to addresses in the form **prefix::1,** with the prefix coming from the sorted allfile.txt. This would typically generate addresses inside the target AS that we would use later as sources after accomplishing the next step to ensure they were source-route capable.

5. Scapy

As mentioned in Chapter 3, Scapy is available for download (BIONDI 2007). Taking the addresses generated in the previous step we would use Scapy to send a probe to that address with a final destination of some other address in an attempt to locate a source-route capable node in the AS. If the appropriate response was received then this

node would given to the probe loader to be used as a source for later probes by the probe engine and alias resolver

F. WRAP UP

System design and implementation have been covered to give a feel of the actual system deployment and its function. In the next chapter, actual system files are detailed and the results of running the system on a live IPv6 AS are covered.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SYSTEM SPECIFIC IMPLEMENTATION AND DEPLOYMENT

A. SYSTEM SPECIFIC IMPLEMENTATION

Detailed in the next few paragraphs are the contents of all the directories of the system and short descriptions of their purpose. All of the code is contained in the appropriately labeled sections of the Appendix of this thesis. All code is written in C and was run on SUSE Linux 10.1.

1. Top Level Directory

This directory contains all the programs created for the system, to include all the test programs that allow off-line testing of the various modules functionality and the actual implementation versions of the system.

Name	Description
agr_stub	Stubs out the generator and receiver section of the alias resolver to allow off-line testing of the alias main function.
alias	The executable for the alias resolver.
alias_offline	The alias resolver but works off-line using the "alias.txt" file created by a previous run of the alias resolver.
anonymous	The executable for the anonymous resolver.
Makefile	Used to direct the compiler as to the source files necessary to generate the executable program. To generate the files needed for deployment type "make". To make all the files, including those needed for testing type "make all". Typing "make clean" will eliminate the object files created for a fresh build in the future.
pgr_stub	Stubs out the generator and receiver section of the probe engine to allow off-line testing of the probe generator function.

probe	The executable for the probing engine
probe_builder	Create all the necessary start-up files for the probe engine.
probe_generator_stub	This is the probe engine with the probe generator stub to allow testing of the probe engine without creating packets.
probe_main_stub	This is the probe generator and receiver with a stub for the main function to allow the testing of the generator and receiver functionality.
probe_receive_stub	This is the probe engine with the probe receiver stub to allow testing of the probe engine without handling responses.

Table 1. – Contents of the top level directory

2. Directory - alias_test_files

This directory contains the needed files to test the alias resolver. These tests are all conducted off-line and there is a README file in the directory detailing how to execute the script.

Name	Description
alias_process_test_file.txt	File to serve as input to the *probe receiver stub* during testing. This will simulate the expected responses to allow the program to resolve some of the nodes.
probe_edges.txt	File containing the initial list of edges to be resolved.
probe_source.txt	File containing the list of sources that will generate the probes.

Table 2. – Contents of the alias_test_files directory

3. Directory - anonymous_test_files

This directory contains the file needed to test the anonymous resolver. Once done the program should eliminate all duplicate threesomes.

Name	Description
probe_edges.txt	Initial file with edges for the *anonymous resolver* to resolve.

Table 3. – Contents of the anonymous_test_files directory

4. Directory - Main_test_files

This directory contains all the scripts and files needed for testing the main function of the probing engine. There is a README file that details how to use each script.

Name	Description
main_probe_process_script1.txt	This contains the proper responses to the *probe receiver stub* when running the *pgr_stub* program and allows the testing of the functioning of the Doubletree algorithm in the main part of the probe engine.
main_probe_process_script2.txt	This contains the proper responses to the *probe receiver stub* when running the *pgr_stub* program and allows the testing of the main function of the program when it hits the source or destination on the first probe.
main_probe_process_script3.txt	This contains the proper responses to the *probe receiver stub* when running the *pgr_stub* program and allows the testing of the main function and its ability to handle hitting a global or local stop.
probe_edges.txt	Required file for start-up of the *pgr_stub* and contains the needed edges for testing.
probe_list.txt	Required file for start-up of the *pgr_stub* and contains the needed nodes remaining to be probed for testing.
probe_source.txt	Required file for start-up of the *pgr_stub* and contains the needed sources for testing.

probe_stop.txt	Required file for start-up of the *pgr_stub* and contains the needed global and local stops for testing.

Table 4. – Contents of the Main_test_files directory

5. Directory – src

This directory contains all the source files needed to create the deployment version of the system. Except for probe_main.h, every implementation file has an associated header file that only contain prototypes of the functions implemented by the module. These header files won't be listed here. Additionally, all the routines use the sys-log system so their relationship to the sys log will not be mentioned explicitly.

Name	Description
alias_main.c	The main function of the alias resolver. This contains the program logic on what modules to execute in what sequence.
alias_main_helper.c	This module implements the fill list, process alias packet, process alias source, generate and resolve detailed in the previous chapter.
alias_offline.c	This is the off-line version of the alias resolver. It contains a majority of the routines needed by the alias resolver. For others it uses those in alias_main_helper.
anon_main.c	This is the main function of the anonymous resolver and is mostly self contained. For those functions it needs externally it calls routines from probe loader, probe ckpnt and probe edges. This routine contains the section called resolver in the previous chapter.
probe_builder.c	This module creates the needed files for the probe engine to start and run. This implements the section called probe builder in the previous chapter.

probe_ckpnt.c	This module contains the routines that dump the in-memory databases to disk creating the files probe_edges.txt, probe_sources.txt, probe_list.txt and probe_stop.txt.
probe_edges.c	This module contains the routines add_edge and add_node which are used to add an edge to the edge list and ensure it is unique. The add node routine checks to see if a node previously has been visited by searching the edge list for its address. This check is used to ensure only new nodes are added to all of the sources' probe list for exploration.
probe_generator.c	This module generates IPv4 encapsulated IPv6 packets based on the IPC message received. It implements the section called packet generator in the previous chapter.
probe_loader.c	This module loads the in-memory databases with the data contained in the files probe_list.txt, probe_edges.txt, probe_sources.txt and probe_edges.txt. This is explained in the checkpoint and loader section in the last chapter.
probe_main.c	The main function of the probe engine. This contains the program logic on what modules to execute in what sequence. This routine implement's the section called main in the probing engine.
probe_main.h	This header file contains all the global variables used, any globally defined constants, and all structures used globally by any of the routines.
probe_main_helper.c	This module contains all the core functions, such as *process source, process packet, process receive, done check, fill list* and *generate*. It also contains the routines insert on list, move, change direction and

	finish with destination.
probe_receive.c	This module listens on the IPv4 address designated by the probe generator for incoming IPv6 encapsulated packets, parses them and sends the resultant information via IPC to the main routine. It implements the section called probe receiver in the previous chapter.
probe_stop.c	This module contains the routines used to check the databases for global and local stops.
probe_utils.c	This module contains routines common to a majority of the other routines used in the system. This also contains the function to send a message to the sys_log.
sys_log.c	This module's only purpose is to create the system log and append all messages it receives to the log. This is explained in the section labeled syslog in the previous chapter.

Table 5. – Contents of the src directory

6. Directory - stubs

This stubs directory contains stub routines to allow users to test the functionality of various routines in the system.

Name	Description
probe_generator_stub.c	This module contains a routine to stub out the generator function for testing purposes. The routine listens to IPC for messages from main and prints out the contents of those messages. It terminates upon a terminate IPC from main.
probe_main_stub.c	This module contains a text-based interface to allow the generation of IPC messages to the receiver and generator. It also interfaces with the *add edge and*

	node and the *global and local stop* routines.
probe_receive_stub.c	This module contains a routine to stub out the receiver function for testing purposes. The routine prompts the user for the required information and generates the proper IPC message to the main routine.

Table 6. – Contents of the stubs directory

7. Directory - Support Programs

Name	Description
my_parse.c	This routine parses data from text versions of RouteViews data. There is no make file for this routine, as a simple call to gcc will build the routine. This function is detailed under parser in the previous chapter.

Table 7. – Contents of the Support Programs directory

B. DEPLOYMENT

The probe engine has been deployed for mapping a real IPv6 AS selected based on RouteViews data. The target AS has over 140 peer ASes and can be considered a large, tier-1 ISP. Mapping a large AS like this turned out to be data intensive.

This section will cover the deployment of each part of the system starting with the probing engine, then the alias resolver, and finally, the anonymous resolver. For each part, we will detail some of the major problems and their resolution and cover some lessons learned.

1. Probing Engine

Simply enter “probe” to see all the command line options for running this program. It should be noted that probe_builder must be run to create the initial startup files in the current directory before running the probe engine. Upon a checkpoint or normal shut-down, the program creates a new copy of the startup files, incorporating the

new information learned during the last cycle, and at the same time, it renames the previous version of the startup files by inserting the time and the day of the week information and changing their suffix to ".bak".

This was by far the most complicated portion of the entire system to implement. The engine required a number of modifications during the first few trial runs to eliminate problems induced by trying to implement the Doubletree algorithm. The largest problem was in the area of anonymous nodes and how to handle all the special cases they created. The first run of the engine produced over 50,000 edges and a total of 40,000 anonymous nodes. A simple change in the program to eliminate the case of starting with an anonymous node eliminated over 15,000 edges and 10,000 anonymous nodes. There were a number of similar cases where a simple change in the engine eliminated a huge number of anonymous nodes.

The single most disappointing part of the entire probing engine was start-up. The engine required over three hours to locate the initial hop count, and then calculate the initial value of *h*. Fortunately, one of the items noticed while correcting for anonymous nodes was that the addition of the ability to source start probes could correct the initial start up problem. It should be considered that the engine is modified to start with an initial value of zero for *h* and the start-up code for determining the value of *h* is eliminated. Considering the poor operation of the start-up code and the ease of which future deployments can correct the problem the number of probes it generated will not be used for evaluating the success or failure of the Doubletree method. Conversely, the most positive point of the engine was having all the check point files in a text format which allowed the review of the data without the creation of any complicated tools and thus bugs in the check point files were eliminated with a simple text editor and the engine could be restarted from the point of the error allowing us to not repeat the entire start-up process again.

Parameter	Value
Packets per Probe	3
Responses Required to be Valid	2
Number of anonymous nodes allowed	1
Timeout in seconds to be considered anonymous	2

Table 8. – Parameters used for probe engine

The final single complete run of the probing engine produced 29,201 edges with 23,280 anonymous nodes and 769 responding nodes using settings in Table 8. On initial glance this looks like an average out-degree for each node of 1.21 which when compared to previous research comes across as highly inaccurate for an AS with 140 peers (Waddington et al. 2003, 59-68). As later learned the way the system produced anonymous nodes was the explanation for this and the problem was corrected with off-line processing. Overall, the engine produced the 29,201 edges using 773,670 packets, or a total of 257,890 probes, each probe contained three packets. Since the engine was using 38 sources and locating 24,049 nodes the author estimates that 913,862 probes (sources * nodes) would have been needed to probe the network using typical probing techniques. While it may seem like only the number of responding nodes, 769 before alias resolution, should only be used for the calculations it is believed that all the sources had to find the anonymous nodes as well and they would be added to the probing load. Therefore it is believed that the Doubletree method resulted in a 72% reduction in the total number of probes.

One of the problems noticed during analysis of the data, and confirmed later by an e-mail from a network administrator, was the engine could not recognize when a source had shut off source-routing. Since testing ran over the course of several weeks, a single administrator in response to our scans terminated source routing on their router. Since the engine had no way of detecting this it continued to hammer the router with requests

without result. So the engine should occasionally check to ensure a source still allows source routing. This is an area recommended for future work.

Overall, the engine performed as expected and with a high degree of success. While the probe engine did raise some eyebrows, over the course of several weeks of tests and during its normal runs it did not seem to cause enough of a stir for any network administrators to contact us. Further, it was not until the run of the alias resolver that problems arose.

One final note, running the engine required the addition of a firewall rule on the host to prevent its protocol stack from answering responses to our packets. The firewall rule for Linux was as follows, “iptables –I OUTPUT 1 –d “ipv4 address” –p icmp –j DROP”. Running the engine without this rule caused a lot of confusion and the addition of numerous unneeded packets.

2. Alias Resolver

Simply enter “alias” to see all the command line options for running this program. It should be noted that the “probe_source.txt” and “probe_edges.txt” files must exist in the current directory before running the alias resolver. The program creates a new “probe_edges.txt” after it renames the original file by inserting the time and the day of the week information and changing the suffix to “.bak”.

This system was the easiest to create but created the largest amount of external problems. Since the resolver sent all the packets to one destination at one time from many sources and every source used a different set of ports it immediately set off network intrusion detection alarms and caused source-routing to be terminated in one case. The resolver led to a temporary suspension of tunnel privileges within 24 hours of its first scan and the tunnel was not reinstated until an explanation was given to some angry administrators. Unfortunately, even slowing down the resolver to 25% of its normal speed, while only raising eyebrows, still set off the alarms so it is recommend this system must be redesigned before use in the future. It is also recommend that any future design be run against an intrusion detection system beforehand to ensure it will not set off alarms.

As for results, this was also the most disappointing of all the parts of the system. The resolver only reduced the 769 original responding nodes to 694, a drop of 75 nodes or a little less than ten percent. Furthermore, once the nodes were resolved the resolver only eliminated 456 edges out of the initial 29201 or less than two percent.

This is the area in which the system could use some large improvements. It is recommended that other methods of alias resolution be found to help reduce these numbers. Unfortunately, future development might result in the use of ICMPv6 quenching; but before that point, it is recommended that some other means be found.

3. Anonymous Resolver

Simply enter "anonymous" to run this program. It should be noted that the "probe_edges.txt" file must exist in the current directory before running the anonymous resolver. The program creates a new "probe_edges.txt" after it renames the original file by inserting the time and the day of the week information and changing the suffix to ".bak".

This part of the system was the most surprising of the whole project. When the system was first run, the number of anonymous nodes found was quite a shock. At first it was thought to be an error in the original probing engine but after correcting the initial coding errors, repeated analysis showed that the file was correct and those 23,280 anonymous nodes were most likely correct. This is when it was realized that this was as large a problem as Yao suggested in his paper (Yao et al. 2003, 353-363). Once the anonymous resolver was run we found and eliminated 3606 edges or 1804 anonymous nodes. While this was a lot more than alias resolution, it still left 25,139 edges for the AS and an out-degree of 1.13 which previous research had shown was certainly not correct (Clauset and Moore 2003).

After more analysis it was found that some initial assumptions caused the probe engine to produce many of those edges incorrectly. While this may seem like an error, it is felt that the engine ran as designed so it was not a coding error but a design error based on an incorrect assumption and this is why it is mentioned here. The first incorrect assumption was when a source had a route prohibited by policy, or for some other reason,

it would drop the packet. In this area even though there was no route the engine would allow that one anonymous node and thus create an edge that was not really there. To correct this, the anonymous resolver was modified to look for these edges and eliminate them. This eliminated 14,793 edges. The second incorrect assumption was that when the probing engine hit a firewall or blocked route it would press on and move an additional two hops, hoping there would be a responding node by the second hop. If there was not a response at the second hop after the firewall it would record an edge from the last responding node to an anonymous node, thus creating an edge to nowhere. The anonymous resolver was modified to remove those false edges and eliminated 6,386 edges. After these two modifications were complete, the edge file contained 3,960 edges for 443 anonymous nodes and 694 resolved nodes and a total of 886 edges containing an anonymous node. This results in an average out-degree of 3.48 which through previous research seems correct for a total of 1137 nodes in an AS peering with more than 140 other ASes (Clauset and Moore 2003).

C. WRAP UP

Overall, this covers the entire probing system and it performance on a live AS. While the engine performed very well it still requires a number of enhancements and corrections which are detailed in the next chapter along with the conclusions.

VI. CONCLUSIONS, RECOMMENDATIONS AND FUTURE WORK

A. CONCLUSION

Overall, the final system seemed to produce a quite believable list of edges from the target AS. Due to time constraints, we were unable to translate the list of edges to a graphical representation and then seek the AS administrator to confirm the accuracy of the resultant map. Even without this, the resultant data seemed consistent with data that would come from such a large AS.

This system was designed as a proof of concept to see if mapping router level topology was feasible considering the differences between IPv4 and IPv6. While there are still some pitfalls, the system mapped an AS while keeping the number of probes to a minimum and successfully accomplished anonymous and alias resolution. The system also proved that source routing is still a viable method to map IPv6 and it ported several router-level topology mapping methods from the IPv4 domain. As explained earlier, the address space size is not a problem for router-level topology but still remains a problem for host level topology. It is felt that the Rocketfuel and Doubletree methodology of mapping are very viable in IPv6 and should be used in the future. The system demonstrated this by its ability to automatically learn new destinations and explore them, while eliminating much of the noise normally generated during this process. While there remains much future work, especially in the area of alias resolution, it is hoped that this will serve as a basis for future research and eventually lead to solutions for the problems remaining.

B. RECOMMENDATIONS AND FUTURE WORK

There are numerous areas in which the system can be improved. This version was intended as a prototype to explore the feasibility pf the concept. It is the initial version. The system should be run in a laboratory environment, again, against an intrusion detection system so that the noise level of the system to an intrusion detection system can

be turned down, especially the alias resolver. The system should be tuned to a level below the normal noise the Internet with respect to the intensity of the probes sent. Of course, doing so will increase the time necessary to map a network topology.

Future research applications of the system should register the tunnel entry point with DNS to supplement the already embedded researcher's e-mail address in the probe header thus allowing administrators of probed networks to see the probes are coming from a legitimate research site and easily contact the researcher if the probes become a problem. Further, posting to IPv6 discussion boards that scans from a certain address are benign and for research. While these may not stop the complaints, and problems could still happen, the operator could state precautions taken limit ill-will and to tell everyone their intentions.

The next few sections detail specific recommendations for improvements and areas for further research. Ultimately, it is recommended that the phases of the system be consolidated so it only one pass limiting the footprint of the system operation on the probed network.

1. Probing Builder

The method of determining the potential probe sources is a manual process and very time intensive. While some tools have been constructed, it is recommended that the entire process be automated enabling the probe engine to accomplish the task.

A possible area of further research is to find new ways to locate sources in the peer ASes. Right now the method detailed is hit and miss and leads to many ASes having no probe points.

2. Probing Engine

The most important fix for the probe engine is to streamline the start-up process. All probes should begin at the source for start up and eliminate the expensive and time consuming routines to determine *h*. As nodes begin to respond, the value of *h* will emerge on its own with the system as it stands and thus it will experience an increasing

benefit of the Doubletree method. Also, the probe engine should only record the valid routes, dropping “routes-to-nowhere” and only record an anonymous node if it eventually connects to another responding node. Next, recommend code be added to deal with route instability and handle it when it happens, right now the engine ignores it.

Another major area of improvement is to have the engine occasionally check if a source is blocking source-routed probes. This would eliminate the injecting of useless packets. Adding a safety check in the probe generator that recognizes when there is an error in the code causing it to send probes repeatedly to the same address that shuts down the system. Would reflect recognition of the importance of guarding against adverse impact on the AS probed. This is recommended because at one point, the probing engine had an endless loop that generated several thousand probes to the same address for no benefit and the safety check would have stopped this error.

An area of possible further research is to add a method to the probe engine to recognize routes that are blocked and stop further probes down that route. This would eliminate many useless probes that are now required to handle this condition.

3. Alias Resolver

This is the area of greatest research potential. Finding new ways to conduct alias resolution is probably the area of greatest need. As current methods only reduce the number of aliased routers by ten percent, resulting maps are bloated and inaccurate.

Another area to be addressed is reducing the likelihood of the resolver setting off intrusion detection alarms. While this seems a simple fix by just slowing the resolver down, the amount of slowdown and the method is not known and needs to be investigated.

4. Anonymous Resolver

The anonymous resolver is another area of great potential in future research. If the recommendations in Yao's paper are incorporated, it is possible that the system could resolve the number of nodes to a handful making the map much more accurate (Yao et al. 2003, 353-363).

Currently, the anonymous resolver runs off-line and takes in the range of minutes to run on 30,000 edges. Unfortunately, as coded it would likely experience an exponential slow down when applied to a greater number of edges. The code should be optimized.

Should these proposed improvements be made, the resulting system may prove very useful for the automatic mapping of an autonomous system for the verification of security design.

APPENDIX A. SCAPY SCRIPTS AND TEST BED CONFIGURATION

Test Case 1:

```
#*****************************************************************************
###############################################################################
#
# This is a script for SCAPY6 Test Case #1
# These will probe the addresses in the standard lab network setup
#
# Author: Robert J. Poulin, Capt, USAF
#
#
###############################################################################
#*****************************************************************************
#
#*****************************************************************************
###############################################################################
# Addresses of all interfaces
###############################################################################
#*****************************************************************************
#
d1='2200:0:0:0255:02::1'
d2='2200:0:0:0255:02::2'
d3='2200:0:0:0250:02::3'
d4='2200:0:0:0250:02::4'
d5='2200:0:0:0240:02::4'
d6='2200:0:0:0240:02::3'
d7='2200:0:0:0244:02::2'
d8='2200:0:0:0244:02::1'
d9='2200:0:0:0222:02::1'
d10='2200:0:0:0222:02::2'
d11='2200:0:0:0220:02::3'
d12='2200:0:0:0220:02::4'
d13='2200:0:0:0210:02::4'
d14='2200:0:0:0210:02::3'
d15='2200:0:0:0211:02::2'
d16='2200:0:0:0211:02::1'
#
###############################################################################
# Addresses that do not exist but are in the address range of the link
###############################################################################
#
d17='2200:0:0:0255:02::55'
d18='2200:0:0:0244:02::55'
d19='2200:0:0:0222:02::55'
d20='2200:0:0:0211:02::55'
#
#*****************************************************************************
###############################################################################
# Setup of static packets
###############################################################################
#*****************************************************************************
#
###############################################################################
# Standard IPv6 Packet
###############################################################################
#
ip=IPv6()
#
###############################################################################
# Standard Routing Header
```

```
###############################################################################
#
route=IPv6ExtHdrRouting()
#
###############################################################################
# Standard ICMPv6 Echo Packet
###############################################################################
#
echo=ICMPv6EchoRequest()
#
###############################################################################
# Standard ICMPv6 EchoRequest
###############################################################################
#
echotest=ip/echo
#
###############################################################################
# Standard TCP Packet no Flags
###############################################################################
#
tcp=TCP(dport=5642,sport=5645)
#
###############################################################################
# Standard UDP Packet no Flags
###############################################################################
#
udp=UDP(dport=5642, sport=5645)
#
###############################################################################
# Standard IPv6 with tcp packet
###############################################################################
#
tcptest=ip/tcp
#
###############################################################################
# Standard IPv6 with udp packet
###############################################################################
#
udptest=ip/udp
#
#*****************************************************************************
###############################################################################
# This next section ensures we have full connectivity by pinging each host
# This should bomb the script if any of the links are down!!
# If this does not run then you should not continue
###############################################################################
#*****************************************************************************
#
###############################################################################
# Test of destination 1
###############################################################################
#
echotest1=echotest
echotest1.dst=d1
rechotest1=sr1(echotest1)
rechotest1.show()
#
###############################################################################
# Test of destination 8
###############################################################################
#
#echotest8=echotest
#echotest8.dst=d8
#rechotest8=sr1(echotest8)
#rechotest8.show()
#
###############################################################################
# Test of destination 9
###############################################################################
```

```
#
echotest9=echotest
echotest9.dst=d9
rechotest9=sr1(echotest9)
rechotest9.show()
#
################################################################################
# Test of destination 16
################################################################################
#
echotest16=echotest
echotest16.dst=d16
rechotest16=sr1(echotest16)
rechotest16.show()
#
#*******************************************************************************
################################################################################
#
# This tests the response of Merlot to source routing packets
# Hop limit is increased by one for each test
#
################################################################################
#*******************************************************************************
#
################################################################################
# From Link Number 1 ICMP
################################################################################
#
################################################################################
# From Link Number 1 ICMP Echo tests
################################################################################
#
echotest5=echotest
echotest5.dst=d5
rechotest5=sr1(echotest5)
rechotest5.show()
echotest13=echotest
echotest13.dst=d13
rechotest13=sr1(echotest13)
rechotest13.show()
#
################################################################################
# Link Number 1 Source Route ICMP echo to d13 hlim = 1
################################################################################
#
route13a=route
route13a.addresses=[d13]
destination13a=ip
destination13a.dst=d5
destination13a.hlim=1
destination13ahalf=destination13a/route13a
final13a=destination13ahalf/echo
rfinal13a=sr(final13a, timeout=5)
#rfinal13a.show()
#
################################################################################
# Link Number 1 Source Route ICMP echo to d13 hlim = 2
################################################################################
#
route13b=route
route13b.addresses=[d13]
destination13b=ip
destination13b.dst=d5
destination13b.hlim=2
destination13bhalf=destination13b/route13b
final13b=destination13bhalf/echo
rfinal13b=sr(final13b, timeout=5)
#rfinal13b.show()
#
```

```
################################################################################
# Link Number 1 Source Route ICMP echo to d13 hlim = 3
################################################################################
#
route13c=route
route13c.addresses=[d13]
destination13c=ip
destination13c.dst=d5
destination13c.hlim=3
destination13chalf=destination13c/route13c
final13c=destination13chalf/echo
rfinal13c=sr(final13c, timeout=5)
#rfinal13c.show()
#
################################################################################
# Link Number 1 Source Route ICMP echo to d13 hlim = 4
################################################################################
#
route13d=route
route13d.addresses=[d13]
destination13d=ip
destination13d.dst=d5
destination13d.hlim=4
destination13dhalf=destination13d/route13d
final13d=destination13dhalf/echo
rfinal13d=sr(final13d, timeout=5)
#rfinal13d.show()
#
################################################################################
# Link Number 1 Source Route ICMP echo to d13 hlim = 5
################################################################################
#
route13e=route
route13e.addresses=[d13]
destination13e=ip
destination13e.dst=d5
destination13e.hlim=5
destination13ehalf=destination13e/route13e
final13e=destination13ehalf/echo
rfinal13e=sr(final13e, timeout=5)
#rfinal13e.show()
#
################################################################################
# From Link Number 3 ICMP
################################################################################
#
################################################################################
# From Link Number 3 ICMP Echo tests
################################################################################
#
route5=route
route5.addresses=[d5]
destination5=ip
destination5.dst=d15
destination5.hlim=64
destination5half=destination5/route5
final5=destination5half/echo
rfinal5=sr(final5, timeout=5)
#rfinal5.show()
route13=route
route13.addresses=[d13]
destination13=ip
destination13.dst=d15
destination13.hlim=64
destination13half=destination13/route13
final13=destination13half/echo
rfinal13=sr(final13, timeout=5)
#rfinal13.show()
#
```

```
################################################################################
# Link Number 3 Source Route ICMP echo to d5 hlim = 1
################################################################################
#
route5f=route
route5f.addresses=[d13,d5]
route5f.segleft=2
destination5f=ip
destination5f.dst=d15
destination5f.hlim=1
destination5fhalf=destination5f/route5f
final5f=destination5fhalf/echo
rfinal5f=sr(final5f, timeout=5)
#rfinal5f.show()
#
################################################################################
# Link Number 3 Source Route ICMP echo to d5 hlim = 2
################################################################################
#
route5g=route
route5g.addresses=[d13,d5]
route5g.segleft=2
destination5g=ip
destination5g.dst=d15
destination5g.hlim=2
destination5ghalf=destination5g/route5g
final5g=destination5ghalf/echo
rfinal5g=sr(final5g, timeout=5)
#rfinal5g.show()
#
################################################################################
# Link Number 3 Source Route ICMP echo to d5 hlim = 3
################################################################################
#
route5h=route
route5h.addresses=[d13,d5]
route5h.segleft=2
destination5h=ip
destination5h.dst=d15
destination5h.hlim=3
destination5hhalf=destination5h/route5h
final5h=destination5hhalf/echo
rfinal5h=sr(final5h, timeout=5)
#rfinal5h.show()
#
################################################################################
# Link Number 3 Source Route ICMP echo to d5 hlim = 4
################################################################################
#
route5i=route
route5i.addresses=[d13,d5]
route5i.segleft=2
destination5i=ip
destination5i.dst=d15
destination5i.hlim=4
destination5ihalf=destination5i/route5i
final5i=destination5ihalf/echo
rfinal5i=sr(final5i, timeout=5)
#rfinal5i.show()
#
################################################################################
# Link Number 3 Source Route ICMP echo to d5 hlim = 5
################################################################################
#
route5j=route
route5j.addresses=[d13,d5]
route5j.segleft=2
destination5j=ip
destination5j.dst=d15
```

```
destination5j.hlim=5
destination5jhalf=destination5j/route5j
final5j=destination5jhalf/echo
rfinal5j=sr(final5j, timeout=5)
#rfinal5j.show()
#
################################################################################
# From Link Number 1 UDP
################################################################################
#
################################################################################
# From Link Number 1 UDP Echo tests
################################################################################
#
udptest5=udptest
udptest5.dst=d5
udptest5=sr1(echotest5)
udptest5.show()
udptest13=udptest
udptest13.dst=d13
udptest13=sr1(echotest13)
udptest13.show()
#
################################################################################
# Link Number 1 Source Route UDP echo to d13 hlim = 1
################################################################################
#
route13k=route
route13k.addresses=[d13]
route13k.segleft=1
destination13k=ip
destination13k.dst=d5
destination13k.hlim=1
destination13khalf=destination13k/route13k
final13k=destination13khalf/udp
rfinal13k=sr(final13k, timeout=5)
#rfinal13k.show()
#
################################################################################
# Link Number 1 Source Route UDP echo to d13 hlim = 2
################################################################################
#
route13l=route
route13l.addresses=[d13]
route13l.segleft=1
destination13l=ip
destination13l.dst=d5
destination13l.hlim=2
destination13lhalf=destination13l/route13l
final13l=destination13lhalf/udp
rfinal13l=sr(final13l, timeout=5)
#rfinal13l.show()
#
################################################################################
# Link Number 1 Source Route UDP echo to d13 hlim = 3
################################################################################
#
route13m=route
route13m.addresses=[d13]
route13m.segleft=1
destination13m=ip
destination13m.dst=d5
destination13m.hlim=3
destination13mhalf=destination13m/route13m
final13m=destination13mhalf/udp
rfinal13m=sr(final13m, timeout=5)
#rfinal13m.show()
#
################################################################################
```

```
# Link Number 1 Source Route UDP echo to d13 hlim = 4
################################################################################
#
route13n=route
route13n.addresses=[d13]
route13n.segleft=1
destination13n=ip
destination13n.dst=d5
destination13n.hlim=4
destination13nhalf=destination13n/route13n
final13n=destination13nhalf/udp
rfinal13n=sr(final13n, timeout=5)
#rfinal13n.show()
#
################################################################################
# Link Number 1 Source Route UDP echo to d13 hlim = 5
################################################################################
#
route13o=route
route13o.addresses=[d13]
route13o.segleft=1
destination13o=ip
destination13o.dst=d5
destination13o.hlim=5
destination13ohalf=destination13o/route13o
final13o=destination13ohalf/udp
rfinal13o=sr(final13o, timeout=5)
#rfinal13o.show()
#
################################################################################
# From Link Number 3 UDP
################################################################################
#
################################################################################
# From Link Number 3 ICMP Echo tests
################################################################################
#
route5p=route
route5p.addresses=[d5]
route5p.segleft=1
destination5p=ip
destination5p.dst=d15
destination5p.hlim=64
destination5phalf=destination5p/route5p
final5p=destination5phalf/udp
rfinal5p=sr(final5p, timeout=5)
#rfinal5p.show()
route13q=route
route13q.addresses=[d13]
route13q.segleft=1
destination13q=ip
destination13q.dst=d15
destination13q.hlim=64
destination13qhalf=destination13q/route13q
final13q=destination13qhalf/udp
rfinal13q=sr(final13q, timeout=5)
#rfinal13q.show()
#
################################################################################
# Link Number 3 Source Route UDP echo to d5 hlim = 1
################################################################################
#
route5r=route
route5r.addresses=[d13,d5]
route5r.segleft=2
destination5r=ip
destination5r.dst=d15
destination5r.hlim=1
destination5rhalf=destination5r/route5r
```

```
final5r=destination5rhalf/udp
rfinal5r=sr(final5r, timeout=5)
#rfinal5r.show()
#
###############################################################################
# Link Number 3 Source Route UDP echo to d5 hlim = 2
###############################################################################
#
route5s=route
route5s.addresses=[d13,d5]
route5s.segleft=2
destination5s=ip
destination5s.dst=d15
destination5s.hlim=2
destination5shalf=destination5s/route5s
final5s=destination5shalf/udp
rfinal5s=sr(final5s, timeout=5)
#rfinal5s.show()
#
###############################################################################
# Link Number 3 Source Route UDP echo to d5 hlim = 3
###############################################################################
#
route5t=route
route5t.addresses=[d13,d5]
route5t.segleft=2
destination5t=ip
destination5t.dst=d15
destination5t.hlim=3
destination5thalf=destination5t/route5t
final5t=destination5thalf/udp
rfinal5t=sr(final5t, timeout=5)
#rfinal5t.show()
#
###############################################################################
# Link Number 3 Source Route UDP echo to d5 hlim = 4
###############################################################################
#
route5u=route
route5u.addresses=[d13,d5]
route5u.segleft=2
destination5u=ip
destination5u.dst=d15
destination5u.hlim=4
destination5uhalf=destination5u/route5u
final5u=destination5uhalf/udp
rfinal5u=sr(final5u, timeout=5)
#rfinal5u.show()
#
###############################################################################
# NOTE: THESE TEST CASES WILL ALSO TEST THE SOURCE ROUTE TEST
###############################################################################
#
#
###############################################################################
# Link Number 3 Source Route UDP echo to d5 hlim = 5
###############################################################################
#
route5v=route
route5v.addresses=[d13,d5]
route5v.segleft=2
destination5v=ip
destination5v.dst=d15
destination5v.hlim=5
destination5vhalf=destination5v/route5v
final5v=destination5vhalf/udp
rfinal5v=sr(final5v, timeout=5)
#rfinal5v.show()
#
```

```
#############################################################################
# Link Number 2 Source Route UDP echo to d10 hlim = 5
#############################################################################
#
route5x=route
route5x.addresses=[d11,d13]
route5x.segleft=2
destination5x=ip
destination5x.dst=d2
destination5x.hlim=120
destination5xhalf=destination5x/route5x
final5x=destination5xhalf/udp
rfinal5x=sr(final5x, timeout=5)
#rfinal5x.show()
#
#############################################################################
# Link Number 2 Source Route UDP echo to d5 hlim = 5
#############################################################################
#
route5w=route
route5w.addresses=[d13,d3]
route5w.segleft=2
destination5w=ip
destination5w.dst=d10
destination5w.hlim=64
destination5whalf=destination5w/route5w
final5w=destination5whalf/udp
rfinal5w=sr(final5w, timeout=5)
#rfinal5w.show()
#
```

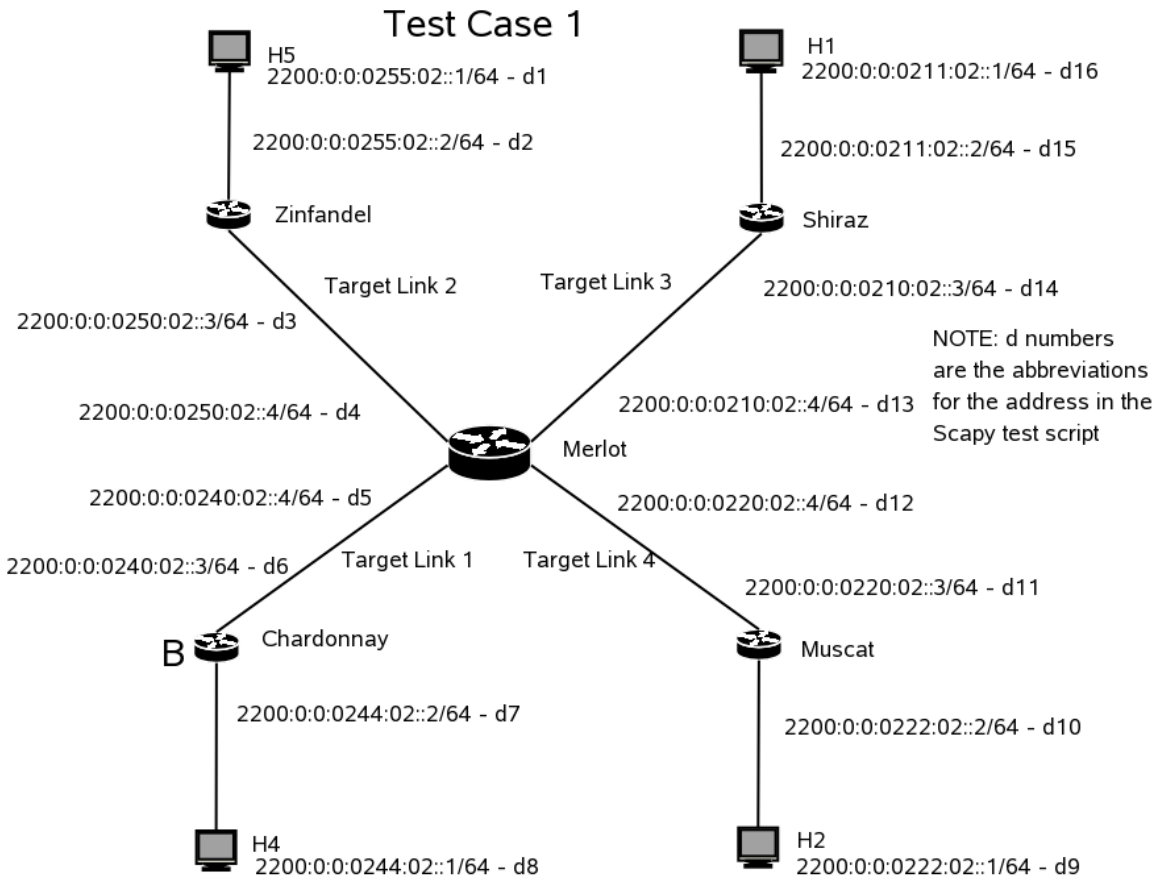

Figure 23 – Lab Test Case 1 Configuration

Test Case 2:

```
#*****************************************************************************
###############################################################################
#
# This is a script for SCAPY6 Test Case #2
# These will probe the addresses in the standard lab network setup
#
# Author: Robert J. Poulin, Capt, USAF
#
#
###############################################################################
#*****************************************************************************
#
#*****************************************************************************
###############################################################################
# Addresses of all interfaces
###############################################################################
#*****************************************************************************
#
d1='2200:0:0:0255:02::1'
d2='2200:0:0:0255:02::2'
d3='2200:0:0:0250:02::3'
d4='2200:0:0:0250:02::4'
d5='2200:0:0:0240:02::4'
d6='2200:0:0:0240:02::3'
d7='2200:0:0:0244:02::2'
d8='2200:0:0:0244:02::1'
d9='2200:0:0:0222:02::1'
d10='2200:0:0:0222:02::2'
d11='2200:0:0:0220:02::3'
d12='2200:0:0:0220:02::4'
d13='2200:0:0:0210:02::4'
d14='2200:0:0:0210:02::3'
d15='2200:0:0:0211:02::2'
d16='2200:0:0:0211:02::1'
#
###############################################################################
# Addresses that do not exist but are in the address range of the link
###############################################################################
#
d17='2200:0:0:0255:02::55'
d18='2200:0:0:0244:02::55'
d19='2200:0:0:0222:02::55'
d20='2200:0:0:0211:02::55'
#
#*****************************************************************************
###############################################################################
# Setup of static packets
###############################################################################
#*****************************************************************************
#
###############################################################################
# Standard IPv6 Packet
###############################################################################
#
ip=IPv6()
#
###############################################################################
# Standard Routing Header
###############################################################################
#
route=IPv6ExtHdrRouting()
#
###############################################################################
# Standard ICMPv6 Echo Packet
###############################################################################
#
echo=ICMPv6EchoRequest()
```

```
#
################################################################################
# Standard ICMPv6 EchoRequest
################################################################################
#
echotest=ip/echo
#
################################################################################
# Standard TCP Packet no Flags
################################################################################
#
tcp=TCP(dport=5642,sport=5645)
#
################################################################################
# Standard UDP Packet no Flags
################################################################################
#
udp=UDP(dport=5642, sport=5645)
#
################################################################################
# Standard IPv6 with tcp packet
################################################################################
#
tcptest=ip/tcp
#
################################################################################
# Standard IPv6 with udp packet
################################################################################
#
udptest=ip/udp
#
#*******************************************************************************
################################################################################
# This next section ensures we have full connectivity by pinging each host
# This should bomb the script if any of the links are down!!
# If this does not run then you should not continue
################################################################################
#*******************************************************************************
#
################################################################################
# Test of destination 1
################################################################################
#
echotest1=echotest
echotest1.dst=d1
rechotest1=sr1(echotest1)
rechotest1.show()
#
################################################################################
# Test of destination 8
################################################################################
#
#echotest8=echotest
#echotest8.dst=d8
#rechotest8=sr1(echotest8)
#rechotest8.show()
#
################################################################################
# Test of destination 9
################################################################################
#
echotest9=echotest
echotest9.dst=d9
rechotest9=sr1(echotest9)
rechotest9.show()
#
################################################################################
# Test of destination 16
################################################################################
```

```
#
echotest16=echotest
echotest16.dst=d16
rechotest16=sr1(echotest16)
rechotest16.show()
#
#*************************************************************************
###########################################################################
#
# ICMP hop limited requests to destinations 3,4,5,11,12,13,14
#
###########################################################################
#*************************************************************************
#
###########################################################################
# Destination 3 ICMP
###########################################################################
#
###########################################################################
# Destination 3 ICMP hlim = 1
###########################################################################
#
echotest3a=echotest
echotest3a.hlim=1
echotest3a.dst=d3
rechotest3a=sr1(echotest3a)
rechotest3a.show()
#
###########################################################################
# Destination 3 ICMP hlim = 2
###########################################################################
#
echotest3b=echotest
echotest3b.hlim=2
echotest3b.dst=d3
rechotest3b=sr1(echotest3b)
rechotest3b.show()
#
###########################################################################
# Destination 3 ICMP hlim = 3
###########################################################################
#
echotest3c=echotest
echotest3c.hlim=1
echotest3c.dst=d3
rechotest3c=sr1(echotest3c)
rechotest3c.show()
#
###########################################################################
# Destination 3 ICMP hlim = 4
###########################################################################
#
echotest3d=echotest
echotest3d.hlim=4
echotest3d.dst=d3
rechotest3d=sr1(echotest3d)
rechotest3d.show()
#
#
###########################################################################
# Destination 3 UDP
###########################################################################
#
###########################################################################
# Destination 3 UDP hlim = 1
###########################################################################
#
udptest3a=udptest
udptest3a.hlim=1
```

```
udptest3a.dst=d3
rudptest3a=sr1(udptest3a)
rudptest3a.show()
#
################################################################################
# Destination 3 UDP hlim = 2
################################################################################
#
udptest3b=udptest
udptest3b.hlim=2
udptest3b.dst=d3
rudptest3b=sr1(udptest3b)
rudptest3b.show()
#
################################################################################
# Destination 3 UDP hlim = 3
################################################################################
#
udptest3c=udptest
udptest3c.hlim=3
udptest3c.dst=d3
rudptest3c=sr1(udptest3c)
rudptest3c.show()
#
################################################################################
# Destination 3 UDP hlim = 4
################################################################################
#
udptest3d=udptest
udptest3d.hlim=4
udptest3d.dst=d3
rudptest3d=sr1(udptest3d)
rudptest3d.show()
#
################################################################################
# Destination 4 ICMP
################################################################################
#
################################################################################
# Destination 4 ICMP hlim = 1
################################################################################
#
echotest4a=echotest
echotest4a.hlim=1
echotest4a.dst=d4
rechotest4a=sr1(echotest4a)
rechotest4a.show()
#
################################################################################
# Destination 4 ICMP hlim = 2
################################################################################
#
echotest4b=echotest
echotest4b.hlim=2
echotest4b.dst=d4
rechotest4b=sr1(echotest4b)
rechotest4b.show()
#
################################################################################
# Destination 4 ICMP hlim = 3
################################################################################
#
echotest4c=echotest
echotest4c.hlim=1
echotest4c.dst=d4
rechotest4c=sr1(echotest4c)
rechotest4c.show()
#
################################################################################
```

```
# Destination 4 ICMP hlim = 4
################################################################################
#
echotest4d=echotest
echotest4d.hlim=4
echotest4d.dst=d4
rechotest4d=sr1(echotest4d)
rechotest4d.show()
#
#
################################################################################
# Destination 4 UDP
################################################################################
#
################################################################################
# Destination 4 UDP hlim = 1
################################################################################
#
udptest4a=udptest
udptest4a.hlim=1
udptest4a.dst=d4
rudptest4a=sr1(udptest4a)
rudptest4a.show()
#
################################################################################
# Destination 4 UDP hlim = 2
################################################################################
#
udptest4b=udptest
udptest4b.hlim=2
udptest4b.dst=d4
rudptest4b=sr1(udptest4b)
rudptest4b.show()
#
################################################################################
# Destination 4 UDP hlim = 3
################################################################################
#
udptest4c=udptest
udptest4c.hlim=3
udptest4c.dst=d4
rudptest4c=sr1(udptest4c)
rudptest4c.show()
#
################################################################################
# Destination 4 UDP hlim = 4
################################################################################
#
udptest4d=udptest
udptest4d.hlim=4
udptest4d.dst=d4
rudptest4d=sr1(udptest4d)
rudptest4d.show()
#
################################################################################
# Destination 11 ICMP
################################################################################
#
################################################################################
# Destination 11 ICMP hlim = 1
################################################################################
#
echotest11a=echotest
echotest11a.hlim=1
echotest11a.dst=d11
rechotest11a=sr1(echotest11a)
rechotest11a.show()
#
################################################################################
```

```
# Destination 11 ICMP hlim = 2
################################################################################
#
echotest11b=echotest
echotest11b.hlim=2
echotest11b.dst=d11
rechotest11b=sr1(echotest11b)
rechotest11b.show()
#
################################################################################
# Destination 11 ICMP hlim = 3
################################################################################
#
echotest11c=echotest
echotest11c.hlim=3
echotest11c.dst=d11
rechotest11c=sr1(echotest11c)
rechotest11c.show()
#
################################################################################
# Destination 11 ICMP hlim = 4
################################################################################
#
echotest11d=echotest
echotest11d.hlim=4
echotest11d.dst=d11
rechotest11d=sr1(echotest11d)
rechotest11d.show()
#
#
################################################################################
# Destination 11 UDP
################################################################################
#
################################################################################
# Destination 11 UDP hlim = 1
################################################################################
#
udptest11a=udptest
udptest11a.hlim=1
udptest11a.dst=d11
rudptest11a=sr1(udptest11a)
rudptest11a.show()
#
################################################################################
# Destination 11 UDP hlim = 2
################################################################################
#
udptest11b=udptest
udptest11b.hlim=2
udptest11b.dst=d11
rudptest11b=sr1(udptest11b)
rudptest11b.show()
#
################################################################################
# Destination 11 UDP hlim = 3
################################################################################
#
udptest11c=udptest
udptest11c.hlim=3
udptest11c.dst=d11
rudptest11c=sr1(udptest11c)
rudptest11c.show()
#
################################################################################
# Destination 11 UDP hlim = 4
################################################################################
#
udptest11d=udptest
```

```
udptest11d.hlim=4
udptest11d.dst=d11
rudptest11d=sr1(udptest11d)
rudptest11d.show()
#
################################################################################
# Destination 12 ICMP
################################################################################
#
################################################################################
# Destination 12 ICMP hlim = 1
################################################################################
#
echotest12a=echotest
echotest12a.hlim=1
echotest12a.dst=d12
rechotest12a=sr1(echotest12a)
rechotest12a.show()
#
################################################################################
# Destination 12 ICMP hlim = 2
################################################################################
#
echotest12b=echotest
echotest12b.hlim=2
echotest12b.dst=d12
rechotest12b=sr1(echotest12b)
rechotest12b.show()
#
################################################################################
# Destination 12 ICMP hlim = 3
################################################################################
#
echotest12c=echotest
echotest12c.hlim=1
echotest12c.dst=d12
rechotest12c=sr1(echotest12c)
rechotest12c.show()
#
################################################################################
# Destination 12 ICMP hlim = 4
################################################################################
#
echotest12d=echotest
echotest12d.hlim=4
echotest12d.dst=d12
rechotest12d=sr1(echotest12d)
rechotest12d.show()
#
#
################################################################################
# Destination 12 UDP
################################################################################
#
################################################################################
# Destination 12 UDP hlim = 1
################################################################################
#
udptest12a=udptest
udptest12a.hlim=1
udptest12a.dst=d12
rudptest12a=sr1(udptest12a)
rudptest12a.show()
#
################################################################################
# Destination 12 UDP hlim = 2
################################################################################
#
udptest12b=udptest
```

```
udptest12b.hlim=2
udptest12b.dst=d12
rudptest12b=sr1(udptest12b)
rudptest12b.show()
#
################################################################################
# Destination 12 UDP hlim = 3
################################################################################
#
udptest12c=udptest
udptest12c.hlim=3
udptest12c.dst=d12
rudptest12c=sr1(udptest12c)
rudptest12c.show()
#
################################################################################
# Destination 12 UDP hlim = 4
################################################################################
#
udptest12d=udptest
udptest12d.hlim=4
udptest12d.dst=d12
rudptest12d=sr1(udptest12d)
rudptest12d.show()
#
################################################################################
# Destination 13 ICMP
################################################################################
#
################################################################################
# Destination 13 ICMP hlim = 1
################################################################################
#
echotest13a=echotest
echotest13a.hlim=1
echotest13a.dst=d13
rechotest13a=sr1(echotest13a)
rechotest13a.show()
#
################################################################################
# Destination 13 ICMP hlim = 2
################################################################################
#
echotest13b=echotest
echotest13b.hlim=2
echotest13b.dst=d13
rechotest13b=sr1(echotest13b)
rechotest13b.show()
#
################################################################################
# Destination 13 ICMP hlim = 3
################################################################################
#
echotest13c=echotest
echotest13c.hlim=1
echotest13c.dst=d13
rechotest13c=sr1(echotest13c)
rechotest13c.show()
#
################################################################################
# Destination 13 ICMP hlim = 4
################################################################################
#
echotest13d=echotest
echotest13d.hlim=4
echotest13d.dst=d13
rechotest13d=sr1(echotest13d)
rechotest13d.show()
#
```

```
#
################################################################################
# Destination 13 UDP
################################################################################
#
################################################################################
# Destination 13 UDP hlim = 1
################################################################################
#
udptest13a=udptest
udptest13a.hlim=1
udptest13a.dst=d13
rudptest13a=sr1(udptest13a)
rudptest13a.show()
#
################################################################################
# Destination 13 UDP hlim = 2
################################################################################
#
udptest13b=udptest
udptest13b.hlim=2
udptest13b.dst=d13
rudptest13b=sr1(udptest13b)
rudptest13b.show()
#
################################################################################
# Destination 13 UDP hlim = 3
################################################################################
#
udptest13c=udptest
udptest13c.hlim=3
udptest13c.dst=d13
rudptest13c=sr1(udptest13c)
rudptest13c.show()
#
################################################################################
# Destination 13 UDP hlim = 4
################################################################################
#
udptest13d=udptest
udptest13d.hlim=4
udptest13d.dst=d13
rudptest13d=sr1(udptest13d)
rudptest13d.show()
#
################################################################################
# Destination 14 ICMP
################################################################################
#
################################################################################
# Destination 14 ICMP hlim = 1
################################################################################
#
echotest14a=echotest
echotest14a.hlim=1
echotest14a.dst=d14
rechotest14a=sr1(echotest14a)
rechotest14a.show()
#
################################################################################
# Destination 14 ICMP hlim = 2
################################################################################
#
echotest14b=echotest
echotest14b.hlim=2
echotest14b.dst=d14
rechotest14b=sr1(echotest14b)
rechotest14b.show()
#
```

```
###############################################################################
# Destination 14 ICMP hlim = 3
###############################################################################
#
echotest14c=echotest
echotest14c.hlim=1
echotest14c.dst=d14
rechotest14c=sr1(echotest14c)
rechotest14c.show()
#
###############################################################################
# Destination 14 ICMP hlim = 4
###############################################################################
#
echotest14d=echotest
echotest14d.hlim=4
echotest14d.dst=d14
rechotest14d=sr1(echotest14d)
rechotest14d.show()
#
#
###############################################################################
# Destination 14 UDP
###############################################################################
#
###############################################################################
# Destination 14 UDP hlim = 1
###############################################################################
#
udptest14a=udptest
udptest14a.hlim=1
udptest14a.dst=d14
rudptest14a=sr1(udptest14a)
rudptest14a.show()
#
###############################################################################
# Destination 14 UDP hlim = 2
###############################################################################
#
udptest14b=udptest
udptest14b.hlim=2
udptest14b.dst=d14
rudptest14b=sr1(udptest14b)
rudptest14b.show()
#
###############################################################################
# Destination 14 UDP hlim = 3
###############################################################################
#
udptest14c=udptest
udptest14c.hlim=3
udptest14c.dst=d14
rudptest14c=sr1(udptest14c)
rudptest14c.show()
#
###############################################################################
# Destination 14 UDP hlim = 4
###############################################################################
#
udptest14d=udptest
udptest14d.hlim=4
udptest14d.dst=d14
rudptest14d=sr1(udptest14d)
rudptest14d.show()
```

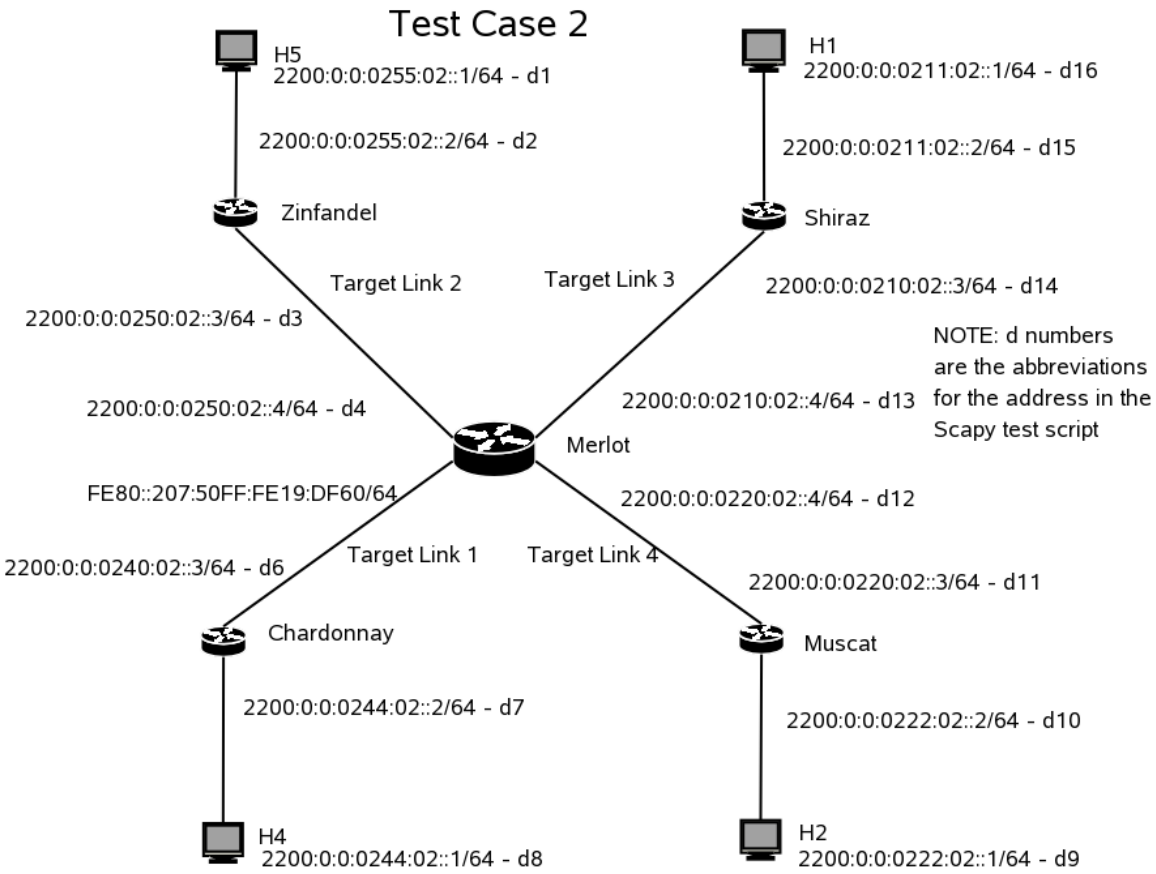

Figure 24 – Lab Test Case 2 Configuration

Test Case 3:

```
#*****************************************************************************
###############################################################################
#
# This is a script for SCAPY6 Test Case #3
# These will probe the addresses in the standard lab network setup
#
# Author: Robert J. Poulin, Capt, USAF
#
#
###############################################################################
#*****************************************************************************
#
#*****************************************************************************
###############################################################################
# Addresses of all interfaces
###############################################################################
#*****************************************************************************
#
d1='2200:0:0:0255:02::1'
d2='2200:0:0:0255:02::2'
d3='2200:0:0:0250:02::3'
d4='2200:0:0:0250:02::4'
d5='2200:0:0:0240:02::4'
d6='2200:0:0:0240:02::3'
d7='2200:0:0:0244:02::2'
d8='2200:0:0:0244:02::1'
d9='2200:0:0:0222:02::1'
d10='2200:0:0:0222:02::2'
d11='2200:0:0:0220:02::3'
d12='2200:0:0:0220:02::4'
d13='2200:0:0:0210:02::4'
d14='2200:0:0:0210:02::3'
d15='2200:0:0:0211:02::2'
d16='2200:0:0:0211:02::1'
d21='2200::2:2ff:2:0:1'
#
###############################################################################
# Addresses that do not exist but are in the address range of the link
###############################################################################
#
d17='2200:0:0:0255:02::55'
d18='2200:0:0:0244:02::55'
d19='2200:0:0:0222:02::55'
d20='2200:0:0:0211:02::55'
#
#*****************************************************************************
###############################################################################
# Setup of static packets
###############################################################################
#*****************************************************************************
#
###############################################################################
# Standard IPv6 Packet
###############################################################################
#
ip=IPv6()
#
###############################################################################
# Standard Routing Header
###############################################################################
#
route=IPv6ExtHdrRouting()
#
###############################################################################
# Standard ICMPv6 Echo Packet
```

```
###############################################################################
#
echo=ICMPv6EchoRequest()
#
###############################################################################
# Standard ICMPv6 EchoRequest
###############################################################################
#
echotest=ip/echo
#
###############################################################################
# Standard TCP Packet no Flags
###############################################################################
#
tcp=TCP(dport=5642,sport=5645)
#
###############################################################################
# Standard UDP Packet no Flags
###############################################################################
#
udp=UDP(dport=5642, sport=5645)
#
###############################################################################
# Standard IPv6 with tcp packet
###############################################################################
#
tcptest=ip/tcp
#
###############################################################################
# Standard IPv6 with udp packet
###############################################################################
#
udptest=ip/udp
#
#******************************************************************************
###############################################################################
# This next section ensures we have full connectivity by pinging each host
# This should bomb the script if any of the links are down!!
# If this does not run then you should not continue
###############################################################################
#******************************************************************************
#
###############################################################################
# Test of destination 1
###############################################################################
#
echotest1=echotest
echotest1.dst=d1
rechotest1=sr1(echotest1)
#rechotest1.show()
#
###############################################################################
# Test of destination 8
###############################################################################
#
#echotest8=echotest
#echotest8.dst=d8
#rechotest8=sr1(echotest8)
#rechotest8.show()
#
###############################################################################
# Test of destination 9
###############################################################################
#
echotest9=echotest
echotest9.dst=d9
rechotest9=sr1(echotest9)
#rechotest9.show()
#
```

```
#############################################################################
# Test of destination 16
#############################################################################
#
echotest16=echotest
echotest16.dst=d16
rechotest16=sr1(echotest16)
#rechotest16.show()
#
#*****************************************************************************
#
#############################################################################
# Destination 2 ICMP
#############################################################################
#
#############################################################################
# Destination 2 ICMP hlim = 1
#############################################################################
#
echotest2a=echotest
echotest2a.hlim=1
echotest2a.dst=d2
rechotest2a=sr1(echotest2a, timeout=5)
#rechotest2a.show()
#
#############################################################################
# Destination 2 ICMP hlim = 2
#############################################################################
#
echotest2b=echotest
echotest2b.hlim=2
echotest2b.dst=d2
rechotest2b=sr1(echotest2b, timeout=5)
#rechotest2b.show()
#
#############################################################################
# Destination 2 ICMP hlim = 3
#############################################################################
#
echotest2c=echotest
echotest2c.hlim=3
echotest2c.dst=d2
rechotest2c=sr1(echotest2c, timeout=5)
#rechotest2c.show()
#
#############################################################################
# Destination 2 ICMP hlim = 4
#############################################################################
#
echotest2d=echotest
echotest2d.hlim=4
echotest2d.dst=d2
rechotest2d=sr1(echotest2d, timeout=5)
#rechotest2d.show()
#
#############################################################################
# Destination 2 UDP
#############################################################################
#
#############################################################################
# Destination 2 UDP hlim = 1
#############################################################################
#
udptest2a=udptest
udptest2a.hlim=1
udptest2a.dst=d2
rudptest2a=sr1(udptest2a, timeout=5)
#rudptest2a.show()
#
```

```
################################################################################
# Destination 2 UDP hlim = 2
################################################################################
#
udptest2b=udptest
udptest2b.hlim=2
udptest2b.dst=d2
rudptest2b=sr1(udptest2b, timeout=5)
#rudptest2b.show()
#
################################################################################
# Destination 2 UDP hlim = 3
################################################################################
#
udptest2c=udptest
udptest2c.hlim=3
udptest2c.dst=d2
rudptest2c=sr1(udptest2c, timeout=5)
#rudptest2c.show()
#
################################################################################
# Destination 2 UDP hlim = 4
################################################################################
#
udptest2d=udptest
udptest2d.hlim=4
udptest2d.dst=d2
rudptest2d=sr1(udptest2d, timeout=5)
#rudptest2d.show()
#
################################################################################
# Destination 10 ICMP
################################################################################
#
################################################################################
# Destination 10 ICMP hlim = 1
################################################################################
#
echotest10a=echotest
echotest10a.hlim=1
echotest10a.dst=d10
rechotest10a=sr1(echotest10a, timeout=5)
#rechotest10a.show()
#
################################################################################
# Destination 10 ICMP hlim = 2
################################################################################
#
echotest10b=echotest
echotest10b.hlim=2
echotest10b.dst=d10
rechotest10b=sr1(echotest10b, timeout=5)
#rechotest10b.show()
#
################################################################################
# Destination 10 ICMP hlim = 3
################################################################################
#
echotest10c=echotest
echotest10c.hlim=1
echotest10c.dst=d10
rechotest10c=sr1(echotest10c, timeout=5)
#rechotest10c.show()
#
################################################################################
# Destination 10 ICMP hlim = 4
################################################################################
#
echotest10d=echotest
```

```
echotest10d.hlim=4
echotest10d.dst=d10
rechotest10d=sr1(echotest10d, timeout=5)
#rechotest10d.show()
#
################################################################################
# Destination 10 UDP
################################################################################
#
################################################################################
# Destination 10 UDP hlim = 1
################################################################################
#
udptest10a=udptest
udptest10a.hlim=1
udptest10a.dst=d10
rudptest10a=sr1(udptest10a, timeout=5)
#rudptest10a.show()
#
################################################################################
# Destination 10 UDP hlim = 2
################################################################################
#
udptest10b=udptest
udptest10b.hlim=2
udptest10b.dst=d10
rudptest10b=sr1(udptest10b, timeout=5)
#rudptest10b.show()
#
################################################################################
# Destination 10 UDP hlim = 3
################################################################################
#
udptest10c=udptest
udptest10c.hlim=3
udptest10c.dst=d10
rudptest10c=sr1(udptest10c, timeout=5)
#rudptest10c.show()
#
################################################################################
# Destination 10 UDP hlim = 4
################################################################################
#
udptest10d=udptest
udptest10d.hlim=4
udptest10d.dst=d10
rudptest10d=sr1(udptest10d, timeout=5)
#rudptest10d.show()
#
################################################################################
# Destination 11 ICMP
################################################################################
#
################################################################################
# Destination 11 ICMP hlim = 1
################################################################################
#
echotest11a=echotest
echotest11a.hlim=1
echotest11a.dst=d11
rechotest11a=sr1(echotest11a, timeout=5)
#rechotest11a.show()
#
################################################################################
# Destination 11 ICMP hlim = 2
################################################################################
#
echotest11b=echotest
echotest11b.hlim=2
```

```
echotest11b.dst=d11
rechotest11b=sr1(echotest11b, timeout=5)
#rechotest11b.show()
#
###############################################################################
# Destination 11 ICMP hlim = 3
###############################################################################
#
echotest11c=echotest
echotest11c.hlim=1
echotest11c.dst=d11
rechotest11c=sr1(echotest11c, timeout=5)
#rechotest11c.show()
#
###############################################################################
# Destination 11 ICMP hlim = 4
###############################################################################
#
echotest11d=echotest
echotest11d.hlim=4
echotest11d.dst=d11
rechotest11d=sr1(echotest11d, timeout=5)
#rechotest11d.show()
#
###############################################################################
# Destination 11 UDP
###############################################################################
#
###############################################################################
# Destination 11 UDP hlim = 1
###############################################################################
#
udptest11a=udptest
udptest11a.hlim=1
udptest11a.dst=d11
rudptest11a=sr1(udptest11a, timeout=5)
#rudptest11a.show()
#
###############################################################################
# Destination 11 UDP hlim = 2
###############################################################################
#
udptest11b=udptest
udptest11b.hlim=2
udptest11b.dst=d11
rudptest11b=sr1(udptest11b, timeout=5)
#rudptest11b.show()
#
###############################################################################
# Destination 11 UDP hlim = 3
###############################################################################
#
udptest11c=udptest
udptest11c.hlim=3
udptest11c.dst=d11
rudptest11c=sr1(udptest11c, timeout=5)
#rudptest11c.show()
#
###############################################################################
# Destination 11 UDP hlim = 4
###############################################################################
#
udptest11d=udptest
udptest11d.hlim=4
udptest11d.dst=d11
rudptest11d=sr1(udptest11d, timeout=5)
#rudptest11d.show()
#
###############################################################################
```

```
# Destination 20 ICMP
################################################################################
#
################################################################################
# Destination 20 ICMP hlim = 1
################################################################################
#
echotest20a=echotest
echotest20a.hlim=1
echotest20a.dst=d20
rechotest20a=sr1(echotest20a, timeout=5)
#rechotest20a.show()
#
################################################################################
# Destination 20 ICMP hlim = 2
################################################################################
#
echotest20b=echotest
echotest20b.hlim=2
echotest20b.dst=d20
rechotest20b=sr1(echotest20b, timeout=5)
#rechotest20b.show()
#
################################################################################
# Destination 20 ICMP hlim = 3
################################################################################
#
echotest20c=echotest
echotest20c.hlim=1
echotest20c.dst=d20
rechotest20c=sr1(echotest20c, timeout=5)
#rechotest20c.show()
#
################################################################################
# Destination 20 ICMP hlim = 4
################################################################################
#
echotest20d=echotest
echotest20d.hlim=4
echotest20d.dst=d20
rechotest20d=sr1(echotest20d, timeout=5)
#rechotest20d.show()
#
################################################################################
# Destination 20 UDP
################################################################################
#
################################################################################
# Destination 20 UDP hlim = 1
################################################################################
#
udptest20a=udptest
udptest20a.hlim=1
udptest20a.dst=d20
rudptest20a=sr1(udptest20a, timeout=5)
#rudptest20a.show()
#
################################################################################
# Destination 20 UDP hlim = 2
################################################################################
#
udptest20b=udptest
udptest20b.hlim=2
udptest20b.dst=d20
rudptest20b=sr1(udptest20b, timeout=5)
#rudptest20b.show()
#
################################################################################
# Destination 20 UDP hlim = 3
```

```
#############################################################################
#
udptest20c=udptest
udptest20c.hlim=3
udptest20c.dst=d20
rudptest20c=sr1(udptest20c, timeout=5)
#rudptest20c.show()
#
#############################################################################
# Destination 20 UDP hlim = 4
#############################################################################
#
udptest20d=udptest
udptest20d.hlim=4
udptest20d.dst=d20
rudptest20d=sr1(udptest20d, timeout=5)
#rudptest20d.show()
#
#############################################################################
# Destination 17 ICMP
#############################################################################
#
#############################################################################
# Destination 17 ICMP hlim = 1
#############################################################################
#
echotest17a=echotest
echotest17a.hlim=1
echotest17a.dst=d17
rechotest17a=sr1(echotest17a, timeout=5)
#rechotest17a.show()
#
#############################################################################
# Destination 17 ICMP hlim = 2
#############################################################################
#
echotest17b=echotest
echotest17b.hlim=2
echotest17b.dst=d17
rechotest17b=sr1(echotest17b, timeout=5)
#rechotest17b.show()
#
#############################################################################
# Destination 17 ICMP hlim = 3
#############################################################################
#
echotest17c=echotest
echotest17c.hlim=1
echotest17c.dst=d17
rechotest17c=sr1(echotest17c, timeout=5)
#rechotest17c.show()
#
#############################################################################
# Destination 17 ICMP hlim = 4
#############################################################################
#
echotest17d=echotest
echotest17d.hlim=4
echotest17d.dst=d17
rechotest17d=sr1(echotest17d, timeout=5)
#rechotest17d.show()
#
#############################################################################
# Destination 17 UDP
#############################################################################
#
#############################################################################
# Destination 17 UDP hlim = 1
#############################################################################
```

```
#
udptest17a=udptest
udptest17a.hlim=1
udptest17a.dst=d17
rudptest17a=sr1(udptest17a, timeout=5)
#rudptest17a.show()
#
###############################################################################
# Destination 17 UDP hlim = 2
###############################################################################
#
udptest17b=udptest
udptest17b.hlim=2
udptest17b.dst=d17
rudptest17b=sr1(udptest17b, timeout=5)
#rudptest17b.show()
#
###############################################################################
# Destination 17 UDP hlim = 3
###############################################################################
#
udptest17c=udptest
udptest17c.hlim=3
udptest17c.dst=d17
rudptest17c=sr1(udptest17c, timeout=5)
#rudptest17c.show()
#
###############################################################################
# Destination 17 UDP hlim = 4
###############################################################################
#
udptest17d=udptest
udptest17d.hlim=4
udptest17d.dst=d17
rudptest17d=sr1(udptest17d, timeout=5)
#rudptest17d.show()
#
###############################################################################
# Destination 21 ICMP
###############################################################################
#
###############################################################################
# Destination 21 ICMP hlim = 1
###############################################################################
#
echotest21a=echotest
echotest21a.hlim=1
echotest21a.dst=d21
rechotest21a=sr1(echotest21a, timeout=5)
#rechotest21a.show()
#
###############################################################################
# Destination 21 ICMP hlim = 2
###############################################################################
#
echotest21b=echotest
echotest21b.hlim=2
echotest21b.dst=d21
rechotest21b=sr1(echotest21b, timeout=5)
#rechotest21b.show()
#
###############################################################################
# Destination 21 ICMP hlim = 3
###############################################################################
#
echotest21c=echotest
echotest21c.hlim=1
echotest21c.dst=d21
rechotest21c=sr1(echotest21c, timeout=5)
```

```
#rechotest21c.show()
#
################################################################################
# Destination 21 ICMP hlim = 4
################################################################################
#
echotest21d=echotest
echotest21d.hlim=4
echotest21d.dst=d21
rechotest21d=sr1(echotest21d, timeout=5)
#rechotest21d.show()
#
#
################################################################################
# Destination 21 UDP
################################################################################
#
################################################################################
# Destination 21 UDP hlim = 1
################################################################################
#
udptest21a=udptest
udptest21a.hlim=1
udptest21a.dst=d21
rudptest21a=sr1(udptest21a, timeout=5)
#rudptest21a.show()
#
################################################################################
# Destination 21 UDP hlim = 2
################################################################################
#
udptest21b=udptest
udptest21b.hlim=2
udptest21b.dst=d21
rudptest21b=sr1(udptest21b, timeout=5)
#rudptest21b.show()
#
################################################################################
# Destination 21 UDP hlim = 3
################################################################################
#
udptest21c=udptest
udptest21c.hlim=3
udptest21c.dst=d21
rudptest21c=sr1(udptest21c, timeout=5)
#rudptest21c.show()
#
################################################################################
# Destination 21 UDP hlim = 4
################################################################################
#
udptest21d=udptest
udptest21d.hlim=4
udptest21d.dst=d21
rudptest21d=sr1(udptest21d, timeout=5)
#rudptest21d.show()
```

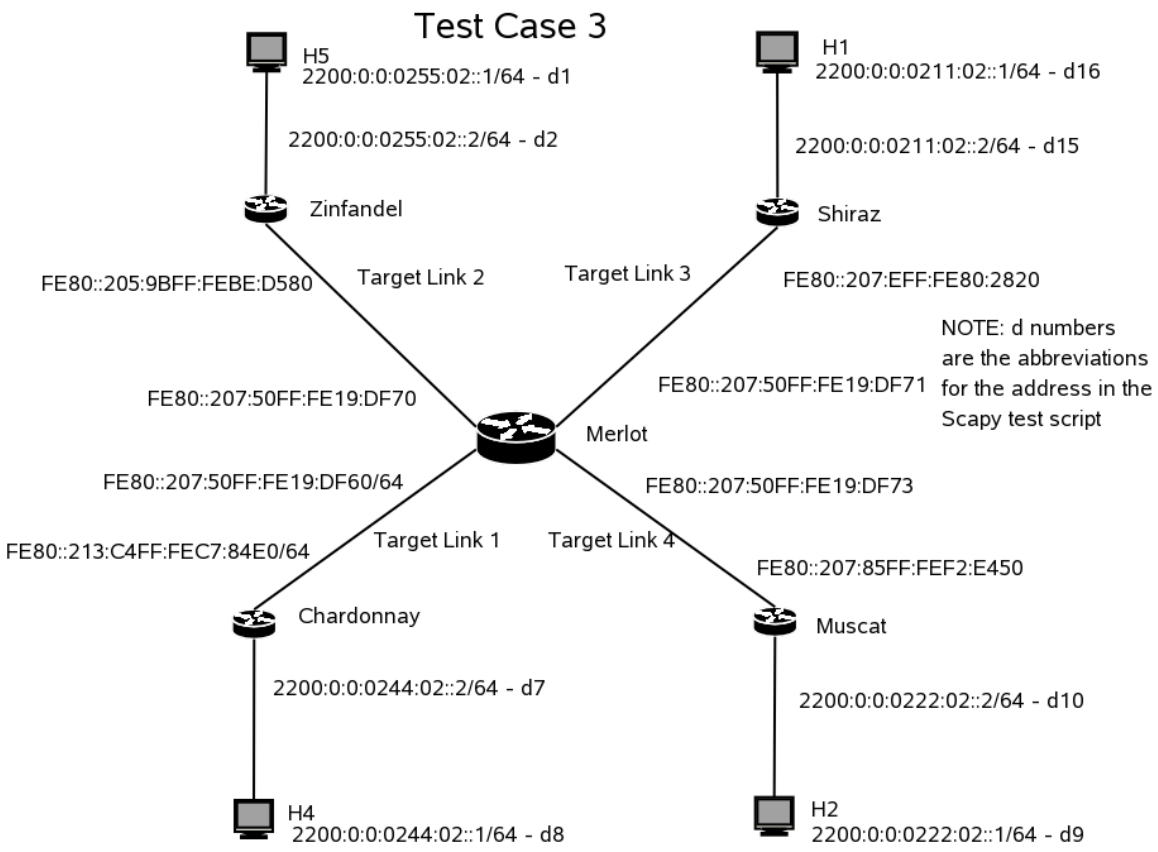

Figure 25 – Lab Test Case 3 Configuration

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. SOURCE CODE

For brevity, only source files for all the modules are included in this appendix, test files are available in the original tar ball. All files are in alphabetical order.

ALIAS MAIN.C

```
/*****************************************************************************
* This is the main program of the alias resolver and controls all operation
*
*
* Author: Robert J. Poulin, Capt, USAF
*
* Program name: alias_main.c
*
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <netinet/in.h>
#include <libnet.h>
#include <signal.h>
#include <unistd.h>
#include <wait.h>
#include "probe_main.h"
#include "probe_generator.h"
#include "probe_receive.h"
#include "probe_utils.h"
#include "probe_loader.h"
#include "probe_ckpnt.h"
#include "alias_main_helper.h"
#include "alias_main.h"
#include "sys_log.h"
#include <sys/socket.h>
#include <arpa/inet.h>
#include <pcap.h>


#define _GNU_SOURCE
#define TROUBLESHOOT 0

/*****************************************************************************
*
* Define constants
*
*****************************************************************************/

   const  int              IPC_ERROR        = -1; //The msg func calls -1 err
   const  int              LIBNET_ERROR     = -1; //Libnet calls -1 an error
   const  int              positive         =  1; //Flg positive direction
   const  int              negative         = -1; //Flg negative direction


/*****************************************************************************
*
* Define externals and globals
*
*****************************************************************************/

   extern int errno;      // Set by calls to the library functions
```

```
   struct generator_buffer  *message_to_generator;  //Buffer for generator msg
   struct receive_buffer    *message_from_receiver; //Buffer for receive msg
   struct sources           *source_list;           //Pointer to list of srcs
   struct edges             *edge_list;             //Pointer to list of edges
   struct global_stops      *global_stop_list;      //Pntr to global stop set
          void*             *probe_source_stop_list;//Pointer to local stop set
   struct alias_nodes       *alias_list;            //Pointer to alias list

   struct messagebuffer     sendbuffer, recvbuffer;
   struct in6_addr          empty;                   //Flag for no address
   struct in6_addr          anonymous_address;       //Addr for anonymous nodes

          int               queueid;                          //Queue id nmb

          char          our_ipv6_addr_name[INET6_ADDRSTRLEN+1];//Our IPv6 addr
          char          tunnelip[INET_ADDRSTRLEN+1];         //Tunnel IPv4 addr
          int           main_done         = PLZCONTINUE;     //Loop stop flg

   int    number_of_sources        = ZERO;   // The number of src for probes
   int    number_of_edges          = ZERO;   // The number of edges found
   int    number_of_global_stops   = ZERO;   // The number of global stops
   int    number_of_nodes          = ZERO;   // The number of total nodes
   int    probe_generator_pid      = ZERO;   // Probe generator PID
   int    sys_log_pid              = ZERO;   // Sys log PID
   int    probe_receive_pid        = ZERO;   // Probe receiver PID
   int    is_main                  = TRUE;   // Tells us if we are main
   time_t check_point_time         = ZERO;   // This records checkpoint time
   unsigned int  anonymous_number  = ZERO;   // The number for anonymous nodes
   int    NUMBER_OF_PACKETS        = ZERO;   // The number of pkts per probe
   int    MIN_RESPONSE             = ZERO;   // Minimum # of response for valid
   int    MAX_ANONYMOUS            = ZERO;   // Needed for probe_main.h
   double DEFAULT_P_VALUE          = 0.0;    // Needed for probe_main.h
   int    WAIT_TIME                = ZERO;   // the time to wait for response
   int    CHECK_POINT_TIME         = ZERO;   // Needed for probe_main.h
   int    deleted_edges            = ZERO;   // Number of edges deleted

/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/

 void main_handler(int sig) {

    int pid     = ZERO;
    int status  = ZERO;

    if (is_main == TRUE) {

       fprintf(stderr, "ALIAS system received the message and is stopping," \
                       " Please wait!\n");
       main_done = STOP;

    }//if (is_main == TRUE) {


}// void main_handler(int sig) {

 void child_handler(int sig) {

    char textbuffer[MAX_MESSAGE];  // Buffer to hold our message
    int  status;
    int  pid         = ZERO;


    pid = wait(&status);
    if (pid == sys_log_pid) {

       fprintf(stderr,"ALIAS MAIN: Sys log stopped, if this is unplanned " \
```

```
                    "check log\n");
        fprintf(stderr,"ALIAS MAIN: The sys log process stopped with status: " \
                    "%i\n", status);
        sys_log_pid = ZERO;
        if (TROUBLESHOOT != 99) {

           main_done   = STOP;     // If sys-log stopped we don't record so stop

        }//if (TROUBLESHOOT != 99) {

     } else if (pid == probe_generator_pid) {

        fprintf(stderr,"ALIAS MAIN: Probe Generator stopped, if this is " \
                    "unplanned check log\n");
        snprintf(textbuffer, MAX_MESSAGE, "ALIAS: The probe generator stopped" \
                                     " with status: %i", status);
        call_sys_log(textbuffer);
        probe_generator_pid = ZERO;
        if (TROUBLESHOOT != 99) {

           main_done = STOP;       // If generator stopped no reason to continue

        }//if (TROUBLESHOOT != 99) {

     } else if (pid == probe_receive_pid) {

        fprintf(stderr,"ALIAS MAIN: Probe Receiver stopped, if this is " \
                    "unplanned check log\n");
        snprintf(textbuffer, MAX_MESSAGE, "ALIAS MAIN: The probe receiver " \
                                     "stopped with status: %i", status);
        call_sys_log(textbuffer);
        probe_receive_pid = ZERO;
        if (TROUBLESHOOT != 99) {

           main_done = STOP;     // If receiver stopped no reason to continue

        }//if (TROUBLESHOOT != 99) {

     } else {

        snprintf(textbuffer, MAX_MESSAGE, "ALIAS MAIN: Received a signal but " \
                                     "pid did not match any running " \
                                     "process, status = : %i\n", status);
        call_sys_log(textbuffer);

     }//if (pid == sys_log_pid) {

}//void child_handler(int sig) {

/*****************************************************************************
 *
 * Main function
 *
 *****************************************************************************/

int main(int argc, char *argv[]){

   /*****************************************************************************
   *
   * Define variables needed by main
   *
   *****************************************************************************/

   struct                   msqid_ds msqid_ds, *buf; //May need this for msg
                                                     //system to check status
                            buf           = &msqid_ds;
   int                      continue_flag = ZERO;    // Flag to tell if any of
                                                     // the processes should
                                                     // continue or quit
```

```
struct in6_addr          destination;             // Used by main to store
                                                  // destination addresses
int                      fork_error    = FALSE;   // Flag to tell us if we
                                                  // had an error forking
int                      forked_pid    = ZERO;    // PID returned by fork
int                      hop_limit     = ZERO;    // Hop limit used by main
key_t                    key           = KEY;     // This will be the value
                                                  // used to determine
                                                  // our queue id
libnet_t                 *packet_handle;          // Initial handle for pckt
int                      payload_type  = ZERO;    // Used by main to store
                                                  // payload type, ICMP or
                                                  // UDP
int                      rcvsuccess    = ZERO;    // value of our receive op
int                      sendsuccess   = ZERO;    // value of our send op
struct in6_addr          source;                  // Used by main to store
                                                  // source addresses
int                      source_port   = ZERO;    // Used by main to store
                                                  // the source port number
char                     textbuffer[MAX_MESSAGE]; // Buffer to hold our
                                                  // messages for sys-log
int                      count         = ZERO;    // General cntr for loops
int                      seconds       = ZERO;    // Seed for random number
int                      processing_done = FALSE; // Ensures we don't save
                                                  // unless we are done

/*****************************************************************************
*
* Set up initial values
*
*****************************************************************************/

time(&seconds); // Get value from system clock and place in seconds for seed
srand((unsigned int) seconds);  //Convert seconds to a unsigned int and seed

int test = inet_pton(AF_INET6, "fe80::0", &empty);
if (test <= ZERO){

   fprintf(stderr,"ALIAS MAIN: Error unable to initialize values " \
                  "\"empty\"\n");
   return -1;

}//if (test <= ZERO){

/*****************************************************************************
*
* Get our IPv6 address and the tunnel endpoint IPv4 address from startup
* If we did not get them on startup, error out and give usage details
*
*****************************************************************************/

if (argc < 6) {

   printf("Alias Program!\n");
   printf("Usage: %s [A] [B] [C] [D] [E] [F]\n",argv[0]);
   printf(" A = Tunnel end point ipv4 address\n");
   printf(" B = Your IPv6 address\n");
   printf(" C = Number of packets per probe\n");
   printf(" D = The minimum number of packets for valid response\n");
   printf(" F = The maximum number of seconds to wait before moving" \
          " to next node\n");
   printf("Example: %s 172.16.0.1 2200::211:2:0:0:2568 3 2 2\n", argv[0]);
   printf("Thanks for playing!!\n");
   return -1;

 }//if (argc < 6) {

 printf("NOTE: IF YOU ARE NOT ROOT THIS WILL FAIL!!!\n");
```

```
 if (TROUBLESHOOT==1) {

    fprintf(stderr, "I am about to do the strncpy\n");
    fprintf(stderr, "The value of argv[1] = %s\n", argv[1]);
    fprintf(stderr, "The value of argv[2] = %s\n", argv[2]);
    fprintf(stderr, "The value of argv[3] = %s\n", argv[3]);
    fprintf(stderr, "The value of argv[4] = %s\n", argv[4]);
    fprintf(stderr, "The value of argv[5] = %s\n", argv[5]);

 }//if (TROUBLESHOOT==1) {

 strncpy(tunnelip, argv[1], INET_ADDRSTRLEN);
 strncpy(our_ipv6_addr_name, argv[2], INET6_ADDRSTRLEN);
 char* endptr              = NULL;    // Used for number conversion
 // Number of packets per transmission
 NUMBER_OF_PACKETS         = (int)  strtoul(argv[3], &endptr, 10);
 // Number of receptions to make it valid
 MIN_RESPONSE              = (int)    strtoul(argv[4], &endptr, 10);
 // the time we will wait for a response
 WAIT_TIME                 = (int)    strtoul(argv[5], &endptr, 10);

 if (TROUBLESHOOT==1) {

   fprintf(stderr, "I am about to do the strncpy\n");
   fprintf(stderr, "The NUMBER_OF_PACKETS on the command line was " \
                   ": %i\n", NUMBER_OF_PACKETS);
   fprintf(stderr, "The MIN_RESPONSE on the command line was : " \
                   "%i\n", MIN_RESPONSE);
   fprintf(stderr, "The WAIT_TIME on the command line was " \
                   ": %i\n", WAIT_TIME);

 }//if (TROUBLESHOOT==1) {

 if (NUMBER_OF_PACKETS <= ZERO) {

    printf("The number of packets is invalid\n");
    return -1;

 } else if ((MIN_RESPONSE > NUMBER_OF_PACKETS) || (MIN_RESPONSE <= ZERO)) {

    printf("The number of responses is invalid\n");
    return -1;

 } else if (WAIT_TIME < ZERO) {

    printf("The number WAIT_TIME is invalid\n");
    return -1;

 }//if (NUMBER_OF_PACKETS <= ZERO) {

 /*************************************************************************
 *
 * Set up the message queue for all to use
 *
 *************************************************************************/

 queueid = msgget(key, 0766 | IPC_CREAT);    // Get our message queue
 if (queueid == IPC_ERROR) {

   fprintf(stderr,"ALIAS MAIN: ERROR: Message Get Failed!!!!\n");
   print_IPC_error("ALIAS MAIN");
   return -1;

} else {

   printf("ALIAS MAIN:Message get succeeded, Queue =  %i\n", queueid );

}//if (queueid == IPC_ERROR) {
```

```
/*************************************************************************
*
* Kick off all the other functions
*
*************************************************************************/

is_main    = TRUE;
fork_error = FALSE;

/*************************************************************************
*
* Kick off syslog
*
*************************************************************************/
forked_pid = fork();

if (forked_pid == ZERO) {

   //We are the child
   is_main = FALSE;
   sys_log(); //We forked successfully we are the child!!!

} else if (forked_pid < ZERO) {

   fprintf(stderr,"ALIAS MAIN: ERROR: trying to fork sys log, return = " \
                  "%i\n", forked_pid);
   fork_error = TRUE;

} else {

   //We are main and everything worked
   sys_log_pid = forked_pid;
   forked_pid  = ZERO;
   if (TROUBLESHOOT == 1){

      fprintf(stderr,"The pid for sys log is: %i\n",sys_log_pid);

   }//if (TROUBLESHOOT == 1){

}//if (forked_pid == ZERO) {

/*************************************************************************
*
* Kick off probe receive
*
*************************************************************************/

if (is_main && fork_error == FALSE) {

   forked_pid = fork();

   if (forked_pid == ZERO) {

      //We are the child
      is_main = FALSE;

      probe_receive(); //We forked successfully we are the child!!!

   } else if (forked_pid < ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: trying to " \
                                          "fork probe receive, return = %i", \
                                          forked_pid);
         call_sys_log(textbuffer);
         fork_error = TRUE;

   } else {

         //We are main and everything worked
```

```
         probe_receive_pid = forked_pid;
         if (TROUBLESHOOT == 1){

            fprintf(stderr,"The pid for probe receive is: %i\n", \
                           probe_receive_pid);

         }//if (TROUBLESHOOT == 1){
         forked_pid  = ZERO;

   }//if (forked_pid == ZERO) {

}//   if (is_main) {

/*****************************************************************************
*
* Kick off probe generator
*
*****************************************************************************/

if (is_main && fork_error == FALSE) {

   forked_pid = fork();

   if (forked_pid == ZERO) {

      //We are the child
      is_main = FALSE;
      probe_generator(); //We forked successfully we are the child!!!

   } else if (forked_pid < ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: trying to " \
                                          "fork probe generator, return =" \
                                          " %i", forked_pid);
         call_sys_log(textbuffer);
         fork_error = TRUE;

   } else {

         //We are main and everything worked
         probe_generator_pid = forked_pid;
         forked_pid  = ZERO;
         if (TROUBLESHOOT == 1){

            fprintf(stderr,"The pid for probe generator is: %i\n", \
                           probe_generator_pid);

         }//if (TROUBLESHOOT == 1){

   }//if (forked_pid == ZERO) {

}//   if (is_main) {

/*****************************************************************************
*
* Setup before we begin probing
*
*****************************************************************************/

if (is_main && fork_error == FALSE) {

   int    startup      = ZERO;        // The success or failure of startup
   int    done         = PLZCONTINUE; // This causes us to loop until we
                                      // are done

   if ((sys_log_pid == ZERO) || (probe_generator_pid == ZERO) ||
      (probe_receive_pid == ZERO)){

      fprintf(stderr, "We had a bad startup, shutting down!!!\n");
```

```
   startup = 2;

}//if ((sys_log_pid == ZERO) || (probe_generator_pid == ZERO) ||

if (startup == ZERO) {

   fprintf(stdout,"Beginning the setup of all the initial values " \
                  "before scanning!\n");
   startup = load_sources();
   if (startup != ZERO) {

      fprintf(stderr, "Error loading sources!\n");

   }//if (startup != ZERO) {

}//if (startup == ZERO) {

if (startup == ZERO) {

   startup = load_nodes();

   if (startup != ZERO) {

      fprintf(stderr, "Error loading nodes!\n");

   } else {

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Total number of " \
                                       "nodes is %i", number_of_nodes);
      call_sys_log(textbuffer);
      fprintf(stdout, "%s\n", textbuffer);

   }//if (startup != ZERO) {

}//if (startup == ZERO) {

/*************************************************************************
*
* Kick off the signal handlers
*
*************************************************************************/

signal(SIGCHLD, child_handler);
signal(SIGQUIT, main_handler);

/*************************************************************************
*
* This is the main routine that accomplishes all the tasks
*
*************************************************************************/

if (TROUBLESHOOT==1) {

   fprintf(stderr,"We are about to setup for the main loop\n");

}//if (TROUBLESHOOT==1) {

if (is_main && (startup == ZERO)) {

   /*********************************************************************
   *
   * Build processing list and initialize it to empty
   *
   *********************************************************************/

   if (TROUBLESHOOT==1) {

      fprintf(stderr,"We are about to empty the processing list\n");
```

```
}//if (TROUBLESHOOT==1) {

struct processing_source processing_list[number_of_sources];
struct processing_source *processing_list_pointer = processing_list;
for(count=ZERO;count<number_of_sources;count++) {

   processing_list[count].destination             = empty;
   processing_list[count].list_number             = ZERO;
   processing_list[count].direction               = ZERO;
   processing_list[count].id                      = ZERO;
   processing_list[count].sequence                = ZERO;
   processing_list[count].hop_count               = ZERO;
   processing_list[count].time_sent               = ZERO;
   processing_list[count].response_number         = ZERO;
   processing_list[count].previous_node           = empty;
   processing_list[count].anonymous_response_count = ZERO;
   processing_list[count].generate                = ZERO;
   processing_list[count].source_start            = FALSE;
   processing_list[count].bad                     = FALSE;

}//for(count=ZERO;count<number_of_sources;count++) {

if (TROUBLESHOOT==1) {

   fprintf(stderr,"We are about to fill the processing list\n");

}//if (TROUBLESHOOT==1) {

fill_alias_list(processing_list);       //Fills the process list

fprintf(stdout,"Done setting up begining the scans!\n");

if (TROUBLESHOOT) {

   fprintf(stderr,"We are about to generate the probes\n");

}//if (TROUBLESHOOT==1) {

generate_alias_probes(processing_list); // Generates probes

fprintf(stdout,"Please enter CTRL \\ to stop!!\n");

if (TROUBLESHOOT) {

   fprintf(stderr,"Entering the main loop\n");

}//if (TROUBLESHOOT==1) {

while (main_done == PLZCONTINUE){

   sleep(WAIT_TIME); // Slows down our processing to prevent floods
   if (main_done == PLZCONTINUE) {

      if (TROUBLESHOOT) {

         fprintf(stderr,"We are about to process alias receive\n");

      }//if (TROUBLESHOOT==1) {

      // Processes all the received packets
      main_done = process_alias_receive(processing_list);

   }//if (main_done == PLZCONTINUE) {

   if (main_done == PLZCONTINUE) {

      if (TROUBLESHOOT) {

         fprintf(stderr,"We are about to fill list\n");
```

```
      }//if (TROUBLESHOOT==1) {

      // Fill the process list
      fill_alias_list(processing_list);

      if (main_done == PLZCONTINUE) {

         if (TROUBLESHOOT) {

            fprintf(stderr,"We are about to done check\n");

         }//if (TROUBLESHOOT==1) {

         // Checks to make sure we are not done
         main_done = alias_done_check(processing_list);
         if (main_done == STOP) {

            processing_done = TRUE;

         }//if (main_done == STOP) {


      }//if (main_done == PLZCONTINUE) {

      if (main_done == PLZCONTINUE) {

         if (TROUBLESHOOT) {

            fprintf(stderr,"We are about to generate\n");

         }//if (TROUBLESHOOT==1) {

         // Generates new probes
         generate_alias_probes(processing_list);

      }//if (main_done == PLZCONTINUE) {

   }//if (main_done = PLZCONTINUE) {

}//while (main_done == PLZCONTINUE){

if (processing_done == TRUE) {

   int update = alias_update();
   if (update == ZERO) {

      if(alias_ckpnt() != ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Failure " \
                                          "updating the edges");
         call_sys_log(textbuffer);
         fprintf(stderr, "ALIAS MAIN: We had a failure updating " \
                         "the edges, check the log for error\n");

      }//if(alias_ckpnt() != ZERO) {

   } else {

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Failure saving " \
                                       "the edges");
      call_sys_log(textbuffer);
      fprintf(stderr, "ALIAS MAIN: We had a failure saving the " \
                      "edges, check the log for error\n");

   }//if (update == ZERO) {

}//if (processing_done == TRUE) {
```

```
} else {

   fprintf(stderr,"ALIAS MAIN: ERROR in loading startup files.  " \
                  "Shutting down!\n");

}//if (startup == ZERO) {

/*************************************************************************
*
* Save if we were successfull
*
*************************************************************************/

if ((startup == ZERO) && (processing_done == TRUE)) {

   int temp = alias_save();
   if (temp != ZERO) {

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Failure saving");
      call_sys_log(textbuffer);
      fprintf(stderr, "ALIAS MAIN: We had a failure saving, check the " \
                      "log for error\n");

   } else {

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Saving successful");
      call_sys_log(textbuffer);
      fprintf(stderr,"%s\n", textbuffer);
      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Deleted %i edges", \
                                       deleted_edges);
      call_sys_log(textbuffer);
      fprintf(stderr,"%s\n", textbuffer);
      int unique_nodes = ZERO;
      for (count=ZERO; count < number_of_nodes; count++) {

         if(cmpaddr(&alias_list[count].original_address,
                    &alias_list[count].resolved_address, 128) == ZERO) {

            unique_nodes++;

         }//if(cmpaddr(&alias_list[count].original_address,

      }//for (count=ZERO; count < number_of_nodes; count++) {
      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Number of original " \
                                       "nodes is %i", number_of_nodes);
      call_sys_log(textbuffer);
      fprintf(stderr,"%s\n", textbuffer);
      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Number of unqiue " \
                                       "nodes left is %i", unique_nodes);
      call_sys_log(textbuffer);
      fprintf(stderr,"%s\n", textbuffer);

   }//if (temp != ZERO) {

}//if (startup == ZERO) {

/*************************************************************************
*
* Kill the probe generator if still running
*
*************************************************************************/

if (probe_generator_pid != ZERO) {

   message_to_generator = (struct generator_buffer*) (sendbuffer.text);
   message_to_generator->generator_cont  = STOP;


  /*************************************************************************
```

```
   *
   * Send the message
   *
   ***********************************************************************/

   sendbuffer.type = GENERATOR;
   sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                        sizeof(struct generator_buffer), IPC_NOWAIT);

   if(sendsuccess != ZERO){

       snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: Message " \
                                        "send failed to stop generator." \
                                        "  Error: ");
       call_sys_log(textbuffer);
       print_IPC_error("ALIAS MAIN");

   } else {

       snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Message was sent " \
                                        "to terminate generator");
       call_sys_log(textbuffer);

   }//if(sendsuccess != ZERO){

}// if (probe_generator_pid != ZERO) {


/***********************************************************************
*
* Wait for all the probes to return before killing the receiver
* NOTE: Loop used instead of sleep(10) because signals wake up sleep
*
***********************************************************************/

int count = 0;
if (probe_receive_pid != ZERO) {

   fprintf(stderr,"ALIAS MAIN: Waiting for 10 seconds for all probes to" \
                  " return before killing probe receive\n");
   while(count <= 10){

      count++;
      sleep(1);

   }//while(count <= 5){

}//if (probe_receive_pid != ZERO) {

if (probe_receive_pid != ZERO) {

 int temp = kill(probe_receive_pid, SIGKILL);
 if (temp < ZERO){

    snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: Message send " \
                                     "failed to stop receiver.  Error: ");
    call_sys_log(textbuffer);
    print_IPC_error("ALIAS MAIN");

 }//if (temp < ZERO){

}//if (probe_receive_pid != ZERO) {

/***********************************************************************
*
* Wait for the probe receive to stop
*
***********************************************************************/
```

```
count = 0;
while((probe_receive_pid  != ZERO) && (count <= 5)){

   fprintf(stderr,"ALIAS MAIN: Waiting for probe receive to finish\n");
   count++;
   sleep(1);

}//while((probe_receive_pid  != ZERO) && (count <= 5)){

/*****************************************************************************
*
* Wait for the probe generator to stop
*
*****************************************************************************/

count = ZERO;
while((probe_generator_pid  != ZERO) && (count <= 5)){

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Waiting for probe " \
                                    "generator to finish");
   printf("%s\n",textbuffer);
   count++;
   sleep(1);

}//while((probe_generator_pid  != ZERO) && (count <= 5)){

/*****************************************************************************
*
* Kill the sys log
*
*****************************************************************************/
sendbuffer.type = SYSLOG;
snprintf(sendbuffer.text, 5, "QUIT");

sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                     sizeof(sendbuffer.text), IPC_NOWAIT);
if(sendsuccess != ZERO){

   fprintf(stderr,"ALIAS MAIN: ERROR: Unable to tell SYS LOG to STOP\n");
   print_IPC_error("ALIAS MAIN");

}//if(sendsuccess != ZERO){

/*****************************************************************************
*
* Wait for the sys log to stop
*
*****************************************************************************/

count = 0;
while((sys_log_pid  != ZERO) && (count <= 5)){

   printf("ALIAS MAIN: Waiting for sys log to finish\n");
   count++;
   sleep(1);

}//while((sys_log_pid  != ZERO) && (count <= 5)){

//Destroy the queue and clean up
int destroy = msgctl(queueid, IPC_RMID, buf);
if(destroy != ZERO){

   fprintf(stderr,"ALIAS MAIN: ERROR: Unable to delete the queue, error" \
                  " number = %i\n", destroy);
   print_IPC_error("ALIAS MAIN");

} else {

   fprintf(stderr,"ALIAS MAIN: Message QUEUE destroyed\n");
```

```
     }//if(destroy != ZERO){
   }// if (is_main && fork_error == FALSE) {
}//int main(int argc, char *argv[])
```

ALIAS_MAIN.H

```
/*****************************************************************************
 * This is the header file of the main alias engine *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: alias_main.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_ALIAS_MAIN
#define INCLUSION_GUARD_PROGRAM_ALIAS_MAIN


#endif //INCLUSION_GUARD_PROGRAM_ALIAS_MAIN
```

ALIAS_MAIN_HELPER.C

```
/*****************************************************************************
 * This is the module contains the helper routines for main to run
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_main_helper.c
 *
 *****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <time.h>
#include <netinet/in.h>
#include "alias_main.h"
#include "probe_main.h"
#include "probe_edges.h"
#include "alias_main_helper.h"
#include "probe_utils.h"
#include "sys_log.h"

#define TROUBLESHOOT 0 // 1 = fill_alias_list      2 = generate_alias_probes
                       // 3 = alias update         4 = process alias packet
                       // 5 = Process alias Source 6 = alias save  7 = resolve
                       // 8 = insert_on_node_list
                       // > 0 = FIXED SRC & DESTINATION PORTS AS DEFINED BELOW
#define ALIASLENGTH  40
#define TEST_DST_PORT 3456
#define TEST_SRC_PORT 2345
#define BASE 10             // Base used for conversion of numbers for strnl

/*****************************************************************************
 *
 * Define externals and globals
 *
 *****************************************************************************/

   extern int errno;      // Set by calls to the library functions to define
                          // errors encountered


          char              textbuffer[MAX_MESSAGE]; // Buffer to hold our
                                                     // message for syslog
                            // Used for testing to increment port
          int               src_port_counter   = TEST_SRC_PORT;
                            // Used for testing to increment port
          int               dst_port_counter   = TEST_DST_PORT;

/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/

  void  process_alias_packet(struct processing_source processing_list[]);
  int   process_alias_source(struct processing_source processing_list[]);
  int   insert_on_list(struct in6_addr address, int finding_source);
  void  pull_from_list(int probe_source, int value_to_pull,
                       struct in6_addr our_address);
  void  clear_from_processing_list(struct processing_source processing_list[],
                                   int probe_source);
  struct in6_addr resolve (struct in6_addr address);
  void  insert_on_node_list(struct in6_addr destination,
                            struct in6_addr response);
```

```
int process_alias_receive(struct processing_source processing_list[]) {

   process_alias_packet(processing_list);
   return process_alias_source(processing_list);

}//void process_alias_receive(struct processing_source *processing_list[]) {


void fill_alias_list(struct processing_source processing_list[]) {

   int count = ZERO;
   int found = FALSE;

   if (TROUBLESHOOT) {

      fprintf(stderr,"We are about to begin filling the list\n");

   }//if (TROUBLESHOOT==1) {

   /*************************************************************************
   *
   * Find the first node that has not been checked.
   *
   *************************************************************************/

   count = ZERO;
   while ((count < number_of_nodes) && (found == FALSE)) {

      if(cmpaddr(&alias_list[count].resolved_address, &empty, 128) == ZERO){

         alias_list[count].resolved_address =
                                       alias_list[count].original_address;
         found = TRUE;

      } else {

         count++;

      }//if(cmpaddr(&alias_list[count].resolved_address, &empty, 128) == ZER

   }//while ((count<number_of_sources) && (found == FALSE)) {


   int the_one = count;

   /*************************************************************************
   *
   * Go through all the sources to see if they are ready for another dst
   *
   *************************************************************************/
   for(count=ZERO; count < number_of_sources; count++) {

      if (TROUBLESHOOT==1) {

         fprintf(stderr,"Working on # %i on the list\n", count);

      }//if (TROUBLESHOOT==1) {


      processing_list[count].response_addr              = empty;
      processing_list[count].direction                  = ZERO;
      processing_list[count].id                         = ZERO;
      processing_list[count].sequence                   = ZERO;
      processing_list[count].hop_count                  = ZERO;
      processing_list[count].time_sent                  = ZERO;
      processing_list[count].response_number            = ZERO;
      processing_list[count].previous_node              = empty;
```

```
      processing_list[count].anonymous_response_count    = ZERO;
      processing_list[count].source_start                = ZERO;
      processing_list[count].bad                         = ZERO;

      if (found == TRUE) {

         processing_list[count].destination =
                                    alias_list[the_one].original_address;
         processing_list[count].list_number                 = the_one;
         processing_list[count].generate                    = TRUE;

      } else {

         processing_list[count].destination                 = empty;
         processing_list[count].list_number                 = ZERO;
         processing_list[count].generate                    = FALSE;

      }//if (found == TRUE) {

   }//for(count=ZERO; count < number_of_sources; count++) {

   if (TROUBLESHOOT==1) {

      fprintf(stderr,"We are done filling the list\n");

   }//if (TROUBLESHOOT==1) {

}//void fill_alias_list() {

void generate_alias_probes(struct processing_source processing_list[]) {

   int count = ZERO;

   /*********************************************************************
   *
   * Go through all the sources to see if they are ready for the next send
   *
   *********************************************************************/

   for(count=ZERO; count < number_of_sources; count++) {

      if(processing_list[count].generate == TRUE) {

         /*****************************************************************
         *
         * If we are here then this source is ready for a new set of packets
         * to be sent
         *
         *****************************************************************/

         struct messagebuffer     sendbuffer;
         struct generator_buffer  *message_to_generator;
         int                      sendsuccess  = ZERO;
        message_to_generator = (struct generator_buffer*) (sendbuffer.text);

         message_to_generator->generator_cont  = PLZCONTINUE;
         message_to_generator->source          = source_list[count].address;
         message_to_generator->destination     =
                                      processing_list[count].destination;
         message_to_generator->hop_limit       =
                               processing_list[count].hop_count + MAX_HOPS;
         message_to_generator->protocol        = SOURCEROUTE + IPPROTO_UDP;
         /*****************************************************************
         *
         * Assign a random id number to this packet
         * NOTE: if troubleshoot fixed port numbers are used asigned above
         *
         *****************************************************************/
```

```
if (TROUBLESHOOT) {

   processing_list[count].id             = src_port_counter;
   processing_list[count].sequence       = dst_port_counter;
   src_port_counter++;
   dst_port_counter++;

} else {

   processing_list[count].id             =
                        rand() % (HIGH - PORT_LOW + 1) + PORT_LOW;
   processing_list[count].sequence       =
                        rand() % (HIGH - PORT_LOW + 1) + PORT_LOW;

}//if (TROUBLESHOOT) {
message_to_generator->port            = processing_list[count].id;
message_to_generator->sequence  = processing_list[count].sequence;

if (TROUBLESHOOT==2) {

   fprintf(stderr, "About to send the message to the generator:" \
                   "\n");
   fprintf(stderr, "The value of generator_cont = %i\n", \
                    message_to_generator->generator_cont);
   char  temp_addr_name[INET6_ADDRSTRLEN+1];
   inet_ntop(AF_INET6, &message_to_generator->source ,
             temp_addr_name, INET6_ADDRSTRLEN);
   fprintf(stderr,"The source address = %s\n", temp_addr_name);
   inet_ntop(AF_INET6, &message_to_generator->destination ,
             temp_addr_name, INET6_ADDRSTRLEN);
   fprintf(stderr, "The destination address = %s\n", \
                    temp_addr_name);
   fprintf(stderr,"The hop limit = %i\n", \
           message_to_generator->hop_limit);
   fprintf(stderr,"The port number = %i\n", \
                  message_to_generator->port);
   fprintf(stderr,"The sequence number = %i\n", \
                  message_to_generator->sequence);

}//if (TROUBLESHOOT==2) {

/*****************************************************************
*
* Send the message
*
*****************************************************************/

sendbuffer.type = GENERATOR;

sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                     sizeof(struct generator_buffer), IPC_NOWAIT);

if(sendsuccess != ZERO){

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: Message " \
                        "send to generator failed.  Error: ");
   call_sys_log(textbuffer);
   print_IPC_error("ALIAS MAIN");

} else {

   /*************************************************************
   *
   * Make sure we don't generate another packet until we are told
   * to and set the anonymous clock to now
   *
   *************************************************************/
   processing_list[count].generate = FALSE;
   processing_list[count].time_sent = time(NULL);
```

```
            char  temp_addr_name[INET6_ADDRSTRLEN+1];
            inet_ntop(AF_INET6, &(processing_list[count].destination) ,
                      temp_addr_name, INET6_ADDRSTRLEN);
            snprintf(textbuffer, MAX_MESSAGE, "ALIAS MAIN: Message sent to" \
                                              " probe generator for dest " \
                                              "%s from source %i", \
                                              temp_addr_name, count);
            call_sys_log(textbuffer);

         }//if(sendsuccess != ZERO){

      }//if(processing_list[count].generate == TRUE) {

   }//for(count=ZERO; count < number_of_sources; count++) {

}//void generate_alias_probes(struct processing_source processing_list[]) {

int  alias_done_check(struct processing_source processing_list[]) {

   /*************************************************************************
   *
   * Check to see if all the sources do not have any remaining destinations
   * if they do not then we are done probing
   *
   *************************************************************************/

   int count = ZERO;

   for(count=ZERO; count < number_of_sources; count++) {

      if(processing_list[count].generate == TRUE) {

         return PLZCONTINUE;

      }//if(processing_list[count].generate = TRUE) {

   }//for(count=ZERO; count < number_of_sources; count++) {

   return STOP;

}//int alias_done_check(struct processing_source processing_list[]) {

int  alias_update() {

  FILE *probe_edges_stream;
  char probe_edges_filename[]  = "probe_edges.txt";  //probe edges file name
  char *in_string              = NULL;               // Pointer to a string
                                                     // for reading the file
  int  length                  = ZERO;               // The number of
                                                     // characters read in
  int  count                   = ZERO;               // Counter used in the
                                                     // loop for reading
  int  check                   = ZERO;   // Value from converting strings to
                                         // binary IPv6 addr

  snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Starting load of edges " \
                                   "from edge file ");
  call_sys_log(textbuffer);

  /*************************************************************************
  *
  * Open the edge file
  *
  *************************************************************************/
  probe_edges_stream = fopen (probe_edges_filename, "r");

  if (probe_edges_stream == NULL) {

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Unable to open " \
```

```
                                    "probe_edges.txt load file");
     call_sys_log(textbuffer);
     return -1;

} else {

   /*********************************************************************
   *
   * This file contains the anonymous node number were working with for
   * use in marking anonymous nodes.  The first number in this file is
   * that number.
   *
   *********************************************************************/

         in_string           = (char *) malloc (MAX_LINE_LENGTH);
   int   temp                = (MAX_LINE_LENGTH - 1);
   char* endptr              = NULL;          // Used for string to number

   if (in_string == NULL){

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: unable to " \
                                       "allocate memory for reading file");
      call_sys_log(textbuffer);
      fclose(probe_edges_stream);
      return -1;

   }//if (in_string == NULL){

   length                = getline(&in_string, &temp, probe_edges_stream);
   if (length <= ZERO) {

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: unable to " \
                           "anonymous node number from probe_edges.txt");
      call_sys_log(textbuffer);
      free(in_string);
      fclose(probe_edges_stream);
      return -1;

   }//if (length <= ZERO) {

   in_string[(length - 1)] = '\0'; //remove any \n at the end of the line
   anonymous_address.in6_u.u6_addr32[0] = htonl(0xfe800000);
   anonymous_address.in6_u.u6_addr32[1] = 0x00000000;
   anonymous_address.in6_u.u6_addr32[2] = 0x00000000;
   anonymous_number = (unsigned int)  strtoul(in_string, &endptr, 16);
   anonymous_address.in6_u.u6_addr32[3] = htonl(anonymous_number);

   if (TROUBLESHOOT==3) {

      fprintf(stderr, "We read in the number %s from the edges file for" \
                      " anonymous number\n", in_string);

   }//if (TROUBLESHOOT==3) {

   if (anonymous_address.in6_u.u6_addr32[3] <= ZERO) {

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: " \
              "probe_edges.txt has incorrect value for anonymous number");
      call_sys_log(textbuffer);
      free(in_string);
      fclose(probe_edges_stream);
      return -1;

   }//if (number_of_edges == ZERO) {

   if (TROUBLESHOOT==3) {

      fprintf(stderr, "We converted the number to %i anonymous number\n",\
                       anonymous_address.in6_u.u6_addr32[3]);
```

```
}//if (TROUBLESHOOT==3) {


/*********************************************************************
*
* Read the number of edge structs in the file
*
*********************************************************************/

length               = getline(&in_string, &temp, probe_edges_stream);
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: unable to " \
                     "read the number of edges from probe_edges.txt");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (length <= ZERO) {

in_string[(length - 1)] = '\0'; //remove any \n at the end of the line
number_of_edges         = (int)  strtoul(in_string, &endptr, BASE);

if (TROUBLESHOOT==3) {

   fprintf(stderr, "We read in the number %s from the edges file\n" \
                  , in_string);

}//if (TROUBLESHOOT==3) {

if (number_of_edges < ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: " \
            "probe_edges.txt has incorrect value for number of edges");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (number_of_edges == ZERO) {

snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Total number of " \
                        "original edges is %i", number_of_edges);
call_sys_log(textbuffer);
fprintf(stdout,"%s\n", textbuffer);

if (TROUBLESHOOT==3) {

   fprintf(stderr, "We converted the number to %i number_of" \
                  "_edges\n", number_of_edges);

}//if (TROUBLESHOOT==3) {

/*********************************************************************
*
* Allocate the memory needed to hold all the node structs
*
*********************************************************************/

struct edges          *current_edge;
       current_edge     = (struct edges*) malloc(sizeof(struct edges));

       edge_list        = (struct edges*) malloc(number_of_edges *
                           sizeof(struct edges));
int    total_edges      = number_of_edges;
       number_of_edges  = ZERO;
```

```
if ((edge_list != NULL) && (current_edge != NULL)) {

   for (count=ZERO; count < total_edges; count++){

      /*****************************************************************
      *
      * Read edge v1
      *
      *****************************************************************/

      length          = getline(&in_string, &temp, probe_edges_stream);
      //remove any \n at the end of the line
      in_string[(length - 1)] = '\0';
      if (length <= ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: problem"\
                                          " reading edge_v1 %i "\
                                          "address from probe_"\
                                          "edges.txt", (count + 1));
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_edges_stream);
         return -1;

      }//if (length <= ZERO) {

      if (TROUBLESHOOT==3) {

         fprintf(stderr, "We read the address %s for count %i\n",\
                          in_string, count);

      }//if (TROUBLESHOOT==3) {

      check = inet_pton(AF_INET6, in_string, &(current_edge->edge_v1));
      if (check <= ZERO){

         snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: problem"\
                  " converting edge_v1 %i address from probe_edges." \
                  "txt", (count + 1));
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_edges_stream);
         return -1;

      }//if (check <= ZERO){

      /*****************************************************************
      *
      * Read edge v2
      *
      *****************************************************************/

      length     = getline(&in_string, &temp, probe_edges_stream);
      //remove any \n at the end of the line
      in_string[(length - 1)] = '\0';
      if (length <= ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: problem"\
                  " reading edge_v2 %i address from probe_edges.txt", \
                  (count + 1));
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_edges_stream);
         return -1;

      }//if (length <= ZERO) {

      if (TROUBLESHOOT==3) {
```

```
   fprintf(stderr, "We read the address %s for count %i\n",\
                    in_string, count);

}//if (TROUBLESHOOT==3) {

check = inet_pton(AF_INET6, in_string, &(current_edge->edge_v2));
if (check <= ZERO){

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: problem"\
            " converting edge_v2 %i address from probe_edges.t"\
            "xt", (count + 1));
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (check <= ZERO){

/*****************************************************************
*
* Read edge starting source
*
*****************************************************************/

length          = getline(&in_string, &temp, probe_edges_stream);
//remove any \n at the end of the line
in_string[(length - 1)] = '\0';
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: problem"\
            " reading starting source %i address from probe_edg"\
            "es.txt", (count + 1));
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (length <= ZERO) {

if (TROUBLESHOOT==3) {

   fprintf(stderr, "We read the address %s for count %i\n", \
                    in_string, count);

}//if (TROUBLESHOOT==3) {

check = inet_pton(AF_INET6, in_string,
                  &(current_edge->starting_source));
if (check <= ZERO){

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: "\
            "problem converting starting source %i addre"\
            "ss from probe_edges.txt", (count + 1));
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (check <= ZERO){

/*****************************************************************
*
* Read edge final dest
*
*****************************************************************/

length          = getline(&in_string, &temp, probe_edges_stream);
//remove any \n at the end of the line
```

```
 in_string[(length - 1)] = '\0';
 if (length <= ZERO) {

    snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: problem"\
            " reading final dest %i address from probe_edges.txt"\
            , (count + 1));
    call_sys_log(textbuffer);
    free(in_string);
    fclose(probe_edges_stream);
    return -1;

 }//if (length <= ZERO) {

 if (TROUBLESHOOT==3) {

    fprintf(stderr, "We read the address %s for count %i\n", \
            in_string, count);

 }//if (TROUBLESHOOT==3) {

 check = inet_pton(AF_INET6, in_string,
                   &(current_edge->final_dest));
 if (check <= ZERO){

    snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: probl"\
             "em converting final dest %i address from probe_e"\
             "dges.txt", (count + 1));
    call_sys_log(textbuffer);
    free(in_string);
    fclose(probe_edges_stream);
    return -1;

 }//if (check <= ZERO){

 /*****************************************************************
 *
 * Read hop count
 *
 *****************************************************************/

 length = getline(&in_string, &temp, probe_edges_stream);
 if (length <= ZERO) {

    snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: unable "\
             "to read edges hop count for %i in probe_edges.txt"\
             , (count + 1));
    call_sys_log(textbuffer);
    free(in_string);
    fclose(probe_edges_stream);
    return -1;

 }//if (length <= ZERO) {

 //remove any \n at the end of the line
 in_string[(length - 1)]   = '\0';
current_edge->hop_count = (int) strtoul(in_string, &endptr, BASE);

 if (TROUBLESHOOT==3) {

    fprintf(stderr, "We read the hop count %s for count %i\n", \
                    in_string, (count+1));

 }//if (TROUBLESHOOT==3) {

 if (current_edge->hop_count < ZERO) {

    snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: probe_"\
             "edges.txt has incorrect value for hop count for %i"\
             , (count + 1));
```

```
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_edges_stream);
         return -1;

      }//if (number_of_edges == ZERO) {

      if (cmpaddr(&current_edge->edge_v1, &empty, 16) != ZERO) {

         current_edge->edge_v1 = resolve(current_edge->edge_v1);

      }//if (cmpaddr(&current_edge->edge_v1, &empty, 16) != ZERO) {

      if (cmpaddr(&current_edge->edge_v2, &empty, 16) != ZERO) {

         current_edge->edge_v2 = resolve(current_edge->edge_v2);

      }//if (cmpaddr(&current_edge->edge_v2, &empty, 16) != ZERO) {

      if(add_edge(current_edge->edge_v1, current_edge->edge_v2,
                  current_edge->starting_source,
                  current_edge->final_dest,
                  current_edge->hop_count ) == NOT_FOUND_AND_INSERTED) {

         if (TROUBLESHOOT==3) {

            fprintf(stderr,"We inserted edge number %i\n", count);
            snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Inserted "\
                     "edge %i", count);
            call_sys_log(textbuffer);

         }//if (TROUBLESHOOT==3) {

      } else {

         if (TROUBLESHOOT==3) {

            fprintf(stderr,"We deleted edge number %i\n", count);

         }//if (TROUBLESHOOT==3) {
         snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: deleted edge"\
                                          " %i", count);
         call_sys_log(textbuffer);
         deleted_edges++;

      }//if(add_edge(current_edge->edge_v1, current_edge->edge_v2,

   }//for (count=ZERO; count < total_edges; count++){

} else {

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: unable to "\
                              "obtain memory for probe_edges.txt");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (edge != NULL) {

free(in_string);
fclose(probe_edges_stream);
snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Finished load and "\
                                 "resolve of edges ");
call_sys_log(textbuffer);
snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Total number of "\
               "edges after alias update is %i", number_of_edges);
call_sys_log(textbuffer);
fprintf(stdout,"%s\n", textbuffer);
```

```
     return ZERO;

  }//if (probe_edges_stream == NULL) {

}//int alias_update()

void process_alias_packet(struct processing_source processing_list[]){

   struct messagebuffer     recvbuffer, sendbuffer;
   //Buffer for messages from receive  function
   struct receive_buffer    *message_from_receiver;
   int                      done         = PLZCONTINUE;
   int                      rcvsuccess   = ZERO;
   int                      sendsuccess  = ZERO;
   int                      probe_source = ZERO;

   /*************************************************************************
   *
   * Process responses until the queue is empty
   *
   *************************************************************************/

   while (done == PLZCONTINUE){

      //Get the message off the queue, do not block
      rcvsuccess = msgrcv(queueid, (void *) &recvbuffer,
                       sizeof(struct receive_buffer), RECEIVE, IPC_NOWAIT);

      if (TROUBLESHOOT==4) {

         fprintf(stderr, "rcvsuccess = %i\n", rcvsuccess);
         if (rcvsuccess == IPC_ERROR) {

            fprintf(stderr, "errno = %i\n", errno);

         }//if (recvsuccess == IPC_ERROR) {

      }//if (TROUBLESHOOT==4) {

      if(rcvsuccess == IPC_ERROR  && errno == ENOMSG) {

         done = STOP;  // Queue is empty exit loop

      }  else if (rcvsuccess == IPC_ERROR && errno != ENOMSG) {

         snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: ERROR: Message "\
                                          "Receive failed.");
         call_sys_log(textbuffer);
         print_IPC_error("ALIAS MAIN");

      } else if (rcvsuccess != IPC_ERROR) {

         message_from_receiver = (struct receive_buffer*) (recvbuffer.text);

         if (TROUBLESHOOT==4) {

            fprintf(stderr, "PROCESS PACKET: The message received is:\n");
            char  temp_addr_name[INET6_ADDRSTRLEN+1];
            inet_ntop(AF_INET6, &message_from_receiver->source ,
                      temp_addr_name, INET6_ADDRSTRLEN);
            fprintf(stderr,"PROCESS PACKET: The received source address = "\
                           "%s\n", temp_addr_name);
            inet_ntop(AF_INET6, &message_from_receiver->destination ,
                      temp_addr_name, INET6_ADDRSTRLEN);
            fprintf(stderr, "PROCESS PACKET: The received destination "\
                            "address = %s\n", temp_addr_name);
            fprintf(stderr,"PROCESS PACKET: The received protocol = %i\n"\
                           , message_from_receiver->protocol);
            inet_ntop(AF_INET6, &message_from_receiver->original_source,
```

```
          temp_addr_name, INET6_ADDRSTRLEN);
 fprintf(stderr,"PROCESS PACKET: The received original source "\
                "address = %s\n", temp_addr_name);
inet_ntop(AF_INET6, &message_from_receiver->original_destination,
           temp_addr_name, INET6_ADDRSTRLEN);
 fprintf(stderr,"PROCESS PACKET: The received original "\
                "destination address = %s\n", temp_addr_name);
 if (message_from_receiver->original_protocol == IPPROTO_ICMPV6){

    fprintf(stderr,"PROCESS PACKET: The received original "\
                   "protocol was ICMPv6\n");

 } else if (message_from_receiver->original_protocol ==
            IPPROTO_UDP) {

    fprintf(stderr,"PROCESS PACKET: The received original pro"\
                   "tocol was UDP\n");

 } else {

    fprintf(stderr,"PROCESS PACKET: The received original "\
                   "protocol did not match it = %i\n", \
                   message_from_receiver->original_protocol);

 }//if (message_from_receiver->original_protocol == IPPROTO_ICMP)
 fprintf(stderr,"PROCESS PACKET: The received original hop "\
                "limit = %i\n", \
                message_from_receiver->original_hop_limit);
 if (message_from_receiver->icmp_type == TIME_EXCEEDED) {

    fprintf(stderr,"PROCESS PACKET: The received icmp type = "\
                   "TIME Exceeded\n");

 } else if (message_from_receiver->icmp_type ==
             DESTINATION_UNREACHABLE) {

    fprintf(stderr,"PROCESS PACKET: The received icmp type = "\
                   "Destination unreachable\n");

 } else {

    fprintf(stderr,"PROCESS PACKET: The received icmp type was"\
         " unknown it = %i\n", message_from_receiver->icmp_type);

 }//if (message_from_receiver->icmp_type == TIME_EXCEEDED) {
 fprintf(stderr,"PROCESS PACKET: The received icmp code = %i\n",\
                 message_from_receiver->icmp_code);
 fprintf(stderr,"PROCESS PACKET: The received icmp ident = %i\n"\
                 , message_from_receiver->icmp_ident);
 fprintf(stderr,"PROCESS PACKET: The received icmp sequence = "\
                "%i\n", message_from_receiver->icmp_sequence);
 fprintf(stderr,"PROCESS PACKET: The received original source "\
                "port = %i\n", \
                message_from_receiver->original_source_port);
 fprintf(stderr,"PROCESS PACKET: The received original dest "\
                "port = %i\n", \
                message_from_receiver->original_dest_port);
 fprintf(stderr,"PROCESS PACKET: The received original route "\
                "seg left = %i\n", \
                message_from_receiver->original_route_seg_left);
 inet_ntop(AF_INET6, &message_from_receiver->original_route_addr,
           temp_addr_name, INET6_ADDRSTRLEN);
 fprintf(stderr,"PROCESS PACKET: The received original route "\
                "address = %s\n", temp_addr_name);
 fprintf(stderr,"PROCESS PACKET: The received original icmp "\
                "type = %i\n", \
                message_from_receiver->original_icmp_type);
 fprintf(stderr,"PROCESS PACKET: The received original icmp "\
                "code = %i\n", \
```

```
                message_from_receiver->original_icmp_code);

 }//if (TROUBLESHOOT==4) {

if ((message_from_receiver->icmp_type == DESTINATION_UNREACHABLE) &&
    (message_from_receiver->icmp_code == PORT_UNREACHABLE)) {

    /*************************************************************
    *
    * Locate the source of this message
    *
    *************************************************************/

     probe_source = ZERO;
     while((processing_list[probe_source].id !=
            message_from_receiver->original_source_port ||
            processing_list[probe_source].sequence !=
             message_from_receiver->original_dest_port||
           (message_from_receiver->original_protocol !=
            IPPROTO_UDP)) &&
            probe_source < number_of_sources){

        if (TROUBLESHOOT==4) {

            fprintf(stderr,"PROCESS PACKET: Attempting locate sour"\
                           "ce of this message comparingin source "\
                           "%i \n", probe_source);
            fprintf(stderr,"PROCESS PACKET: The processing_list[pr"\
                           "obe_source].id = %i\n", \
                            processing_list[probe_source].id);
            fprintf(stderr,"PROCESS PACKET: The message_from_recei"\
                           "ver->original_source_port = %i\n", \
                           message_from_receiver->original_source_port);
            fprintf(stderr,"PROCESS PACKET: The message_from_recei"\
                           "ver->icmp_ident = %i\n", \
                           message_from_receiver->icmp_ident);
            fprintf(stderr,"PROCESS PACKET: The processing_list[pr"\
                           "obe_source].sequence = %i\n", \
                           processing_list[probe_source].sequence);
            fprintf(stderr,"PROCESS PACKET: The message_from_recei"\
                           "ver->original_dest_port = %i\n", \
                            message_from_receiver->original_dest_port);
            fprintf(stderr,"PROCESS PACKET: The message_from_recei"\
                           "ver->icmp_sequence = %i\n", \
                            message_from_receiver->icmp_sequence);
            char  temp_addr_name[INET6_ADDRSTRLEN+1];
        inet_ntop(AF_INET6, &message_from_receiver->original_source,
                  temp_addr_name, INET6_ADDRSTRLEN);
            fprintf(stderr,"PROCESS PACKET: The message_from_recei"\
                           "ver->original_source = %s\n",\
                           temp_addr_name);
            inet_ntop(AF_INET6, &source_list[probe_source].address,
                      temp_addr_name, INET6_ADDRSTRLEN);
            fprintf(stderr,"PROCESS PACKET: The source_list[probe_"\
                          "source].address = %s\n", temp_addr_name);
            fprintf(stderr,"PROCESS PACKET: The message_from_recei"\
                           "ver->protocol = %i\n", \
                           message_from_receiver->protocol);
            fprintf(stderr,"PROCESS PACKET: The IPPROTO_ICMPV6 ="\
                           " %i\n", IPPROTO_ICMPV6);

        }//if (TROUBLESHOOT==4) {

        probe_source++;

     }//while(processing_list[probe_source].id != message_from_recei

     if (probe_source >= number_of_sources) {
```

```
    /*********************************************************
    *
    * If we are here the packet is not from the set we are
    * working on now
    *
    *********************************************************/

    char  temp_addr_name[INET6_ADDRSTRLEN+1];
    inet_ntop(AF_INET6, &message_from_receiver->source,
              temp_addr_name, INET6_ADDRSTRLEN);
    snprintf(textbuffer, MAX_MESSAGE, "ALIAS MAIN: Received "\
             "unknown packet from %s source", temp_addr_name);
    call_sys_log(textbuffer);
    if (TROUBLESHOOT==4) {

       fprintf(stderr,"%s\n", textbuffer);

    }//if (TROUBLESHOOT==4) {

 } else {

    /*********************************************************
    *
    * Add the packet to the processing list as a received packet
    *
    *********************************************************/

    if (cmpaddr(&processing_list[probe_source].response_addr,
                &empty, 128) == ZERO){

       /*******************************************************
       *
       * If we are here then this is the first response and we
       * need to load the address of the response
       *
       *******************************************************/

       processing_list[probe_source].response_addr =
                                    message_from_receiver->source;
       processing_list[probe_source].response_number++;

} else if (cmpaddr(&processing_list[probe_source].response_addr,
                   &message_from_receiver->source, 128) == ZERO){

       /*******************************************************
       *
       * If we are here then this is the not the first and we
       * just increment the count
       *
       *******************************************************/

       processing_list[probe_source].response_number++;

    } else {

       /*******************************************************
       *
       * If we are here then we have received a response from
       * more than one destination
       * to the three probes
       *
       *******************************************************/

       char  temp_addr_name[INET6_ADDRSTRLEN+1];
inet_ntop(AF_INET6, &processing_list[probe_source].destination,
         temp_addr_name, INET6_ADDRSTRLEN);
       snprintf(textbuffer, MAX_MESSAGE, "ALIAS MAIN: Received "\
                "packet from two sources for destination %s, "\
                "hop count %i", temp_addr_name, \
```

```
                              processing_list[probe_source].hop_count);
                     call_sys_log(textbuffer);

                    }//if (cmpaddr(&processing_list[probe_source].response_addr,

                  }//if (probe_source >= number_of_sources) {

               } else {

                  char  temp_addr_name[INET6_ADDRSTRLEN+1];
                  inet_ntop(AF_INET6, &message_from_receiver->source ,
                            temp_addr_name, INET6_ADDRSTRLEN);
                  snprintf(textbuffer, MAX_MESSAGE, "ALIAS MAIN: Rcvd packet "\
                           "from %s src with type % i and code % i", \
                           temp_addr_name, message_from_receiver->icmp_type, \
                           message_from_receiver->icmp_code);
                  call_sys_log(textbuffer);

                  if (TROUBLESHOOT==4) {

                    fprintf(stderr,"PROCESS PACKET: %s!\n",textbuffer);

                  }//if (TROUBLESHOOT==4) {

               }//if ((message_from_receiver->icmp_type == DESTINATION_UNREACHABLE

            }//if(rcvsuccess == IPC_ERROR  && errno == ENOMSG) {

         }//while (done == PLZCONTINUE){

      }//void process_alias_packet(struct processing_source processing_list[]){

      int process_alias_source(struct processing_source processing_list[]){

        int probe_source = ZERO;

        for (probe_source=ZERO;probe_source<number_of_sources;probe_source++) {

           if (TROUBLESHOOT==5) {

              fprintf(stderr,"ALIAS Source %i has %i responses\n", (probe_source)\
                             , processing_list[probe_source].response_number);

           }//if (TROUBLESHOOT==5) {

           if (processing_list[probe_source].response_number >= MIN_RESPONSE) {

              if (cmpaddr(&processing_list[probe_source].destination,
                          &processing_list[probe_source].response_addr, 128) !=
                           ZERO) {

                 insert_on_node_list(processing_list[probe_source].destination,
                                     processing_list[probe_source].response_addr);
                 clear_from_processing_list(processing_list, probe_source);

              }//if (cmpaddr(&processing_list[probe_source].destination, processin

           }//if (processing_list[probe_source].response_number >= MIN_RESPONSE) {

        }//for (probe_source=ZERO;probe_source<number_of_sources;probe_source++) {

        return PLZCONTINUE;

      }//int process_alias_source(struct processing_source *processing_list[]){


/*****************************************************************************
 *
 * Subroutines
```

```
*
*****************************************************************************/

  int alias_save(void) {

     time_t                   curtime;    // The current time in system format
     struct tm                *loctime;
     char                     date[DATELENGTH]; //Used to hold current date
     // Pointer to a string for writing to the file
     char                     *out_string                  = NULL;
     // Counter used in the loop for reading
     int                      count                        = ZERO;
     FILE                     *alias_stream;
     // name of the alias file
     char                     alias_filename[]             = "alias.txt";
     // name of the alias backup file
     char                     alias_filebak[ALIASLENGTH];


     curtime = time(NULL);

     //Convert it it to local time representation.
     loctime = localtime(&curtime);
     int convert = strftime(date, DATELENGTH, "%H%M%a",loctime);
     if (convert != (7)) {

        snprintf(textbuffer, MAX_MESSAGE,"ALIAS SAVE: PROBLEM STARTING, could"\
                                        " not get date for file name");
        call_sys_log(textbuffer);
        return -1;

     }//if (num != (7)) {

     snprintf(alias_filebak, ALIASLENGTH,"alias%s.bak", date);

     snprintf(textbuffer, MAX_MESSAGE,"ALIAS SAVE: Starting the save of "\
                                     "alias file");
     call_sys_log(textbuffer);

     /*************************************************************************
     *
     * Back up old alias file
     *
     *************************************************************************/

     int ren = rename(alias_filename, alias_filebak);

     if ((ren < ZERO) && (errno != NO_ORIGINAL_FILE)) {

         snprintf(textbuffer, MAX_MESSAGE,"ALIAS SAVE: Unable to make alias "\
                                         "backup file! errno = %i", errno);
         call_sys_log(textbuffer);
         return -1;

     } else if (errno == NO_ORIGINAL_FILE) {

         snprintf(textbuffer, MAX_MESSAGE,"ALIAS SAVE: No original file to "\
                                         "backup for alias.txt");
         call_sys_log(textbuffer);
         if (TROUBLESHOOT==6) {

            fprintf(stderr, "%s\n", textbuffer);

         }//if (TROUBLESHOOT==6) {

     } else {

        if (TROUBLESHOOT==6) {
```

```
      fprintf(stderr, "We backed up %s to %s \n", alias_filename,\
                      alias_filebak);

   }//if (TROUBLESHOOT==6) {

}//if (ren < ZERO) {

/*********************************************************************
*
* Create the space for the ourstring
*
*********************************************************************/

out_string          = (char *) malloc (MAX_LINE_LENGTH);

if (out_string == NULL){

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS SAVE: ERROR: unable to "\
                                    "allocate memory for strings");
   call_sys_log(textbuffer);
   return -1;

}//if (in_string == NULL){

/*********************************************************************
*
* Open the alias file
*
*********************************************************************/
alias_stream = fopen (alias_filename, "w");

if (alias_stream == NULL) {

    snprintf(textbuffer, MAX_MESSAGE,"ALIAS SAVE: Unable to open "\
                                     "alias.txt file");
    call_sys_log(textbuffer);
    return -1;

} else {

   /*********************************************************************
   *
   * Write the number of aliases in the file
   *
   *********************************************************************/

   fprintf (alias_stream, "%i\n", number_of_nodes);

   if (TROUBLESHOOT==6) {

      fprintf(stderr, "We wrote number_of_nodes %i to the alias file: "\
                      "%s\n", number_of_nodes, alias_filename);

   }//if (TROUBLESHOOT==6) {

   /*********************************************************************
   *
   * Write the aliases
   *
   *********************************************************************/

   for (count=ZERO; count < number_of_nodes; count++){

      /*********************************************************************
      *
      * write original address
      *
      *********************************************************************/
```

```
if (TROUBLESHOOT==6) {

   fprintf(stderr, "About to convert!\n");

}//if (TROUBLESHOOT==6) {

if (inet_ntop(AF_INET6, &(alias_list[count].original_address),
              out_string, INET6_ADDRSTRLEN) == NULL){

   if (TROUBLESHOOT==6) {

      fprintf(stderr, "convert done!\n");

   }//if (TROUBLESHOOT==6) {

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS SAVE: ERROR: problem "\
            "converting original %i address for alias.txt", \
            (count + 1));
   call_sys_log(textbuffer);
   free(out_string);
   fclose(alias_stream);
   return -1;

}//if (inet_ntop(AF_INET6, &(alias_list[count].original_address) ,

fprintf (alias_stream, "%s\n", out_string);

if (TROUBLESHOOT==6) {

   fprintf(stderr, "We wrote original address %s to the alias fil"\
                   "e: %s\n", out_string, alias_filename);

}//if (TROUBLESHOOT==6) {

/*****************************************************************
*
* write resolved address
*
*****************************************************************/

if (TROUBLESHOOT==6) {

   fprintf(stderr, "About to convert!\n");

}//if (TROUBLESHOOT==6) {

if (inet_ntop(AF_INET6, &(alias_list[count].resolved_address),
    out_string, INET6_ADDRSTRLEN) == NULL){

   if (TROUBLESHOOT==6) {

      fprintf(stderr, "convert done!\n");

   }//if (TROUBLESHOOT==6) {

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS SAVE: ERROR: problem "\
            "converting resolved %i address for alias.txt", \
            (count + 1));
   call_sys_log(textbuffer);
   free(out_string);
   fclose(alias_stream);
   return -1;

}//if (inet_ntop(AF_INET6, &(alias_list[count].resolved_address) ,

fprintf (alias_stream, "%s\n", out_string);

if (TROUBLESHOOT==6) {
```

```
            fprintf(stderr, "We wrote resolved address %s to the alias file"\
                            ": %s\n", out_string, alias_filename);

         }//if (TROUBLESHOOT==6) {

      }//for (count=ZERO; count < number_of_sources; count++){

      free(out_string);
      fclose(alias_stream);

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS SAVE: Finished saving "\
                                       "addresses ");
      call_sys_log(textbuffer);
      return ZERO;

   }//if (alias_stream == NULL) {

}//int alias_save(void) {

void clear_from_processing_list(struct processing_source processing_list[],
                                int probe_source){

   /*********************************************************************
   *
   * This is to make the code mode readable, it just clears this destination
   * frpm the processing list
   *
   *********************************************************************/

   processing_list[probe_source].destination              = empty;
   processing_list[probe_source].response_addr            = empty;
   processing_list[probe_source].list_number              = ZERO;
   processing_list[probe_source].direction                = ZERO;
   processing_list[probe_source].id                       = ZERO;
   processing_list[probe_source].sequence                 = ZERO;
   processing_list[probe_source].hop_count                = ZERO;
   processing_list[probe_source].time_sent                = ZERO;
   processing_list[probe_source].response_number          = ZERO;
   processing_list[probe_source].previous_node            = empty;
   processing_list[probe_source].anonymous_response_count = ZERO;
   processing_list[probe_source].generate                 = FALSE;
   processing_list[probe_source].source_start             = FALSE;
   processing_list[probe_source].bad                      = FALSE;

}//void clear_from_processing_list(struct processing_source processing_list


struct in6_addr resolve (struct in6_addr address) {

  int middle = ZERO;  // Index into the global array to locate the item
  int bottom = ZERO;
  int result = ZERO;  // The result of a comparison
  int count  = ZERO;  // Counter for loops
  int top    = (number_of_nodes  - 1);

  if (TROUBLESHOOT==7) {

     fprintf(stderr, "We are starting resolve\n");

  }//if (TROUBLESHOOT==7) {

  while (bottom <= top) {

     middle = (bottom + top) / 2;
     result = cmpaddr(&address, &(alias_list[middle].original_address),
                      128);
     if (TROUBLESHOOT==7) {

       fprintf(stderr, "The value of bottom = %i\n", bottom);
```

```
        fprintf(stderr, "The value of top    = %i\n", top);
        fprintf(stderr, "The value of middle = %i\n", middle);
        fprintf(stderr, "The value of result = %i\n", result);

      }//if (TROUBLESHOOT==7) {

      if (result == ZERO){

         /*****************************************************************
         *
         * If we are here we found the node
         *
         *****************************************************************/

         return alias_list[middle].resolved_address;

      } else if (result < ZERO) {

         top = middle - 1;

      } else {

         bottom = middle + 1;

      }//if (result == ZERO){

   }//while (bottom <= top) {

   return address;

}//struct in6_addr resolve (struct in6_addr address) {


void insert_on_node_list(struct in6_addr destination,
                         struct in6_addr response) {

   int middle = ZERO;  // Index into the global array to locate the item
   int bottom = ZERO;
   int result = 1;     // The result of a comparison, set to one to allow
                       // initial entry into while loop
   int count  = ZERO;  // Counter for loops
   int top    = (number_of_nodes  - 1);

   if (TROUBLESHOOT==8) {

      fprintf(stderr, "We are starting insert_on_node_list\n");
      char  temp_addr_name[INET6_ADDRSTRLEN+1];
      char  temp_addr_name1[INET6_ADDRSTRLEN+1];
      inet_ntop(AF_INET6, &destination , temp_addr_name, INET6_ADDRSTRLEN);
      inet_ntop(AF_INET6, &response , temp_addr_name1, INET6_ADDRSTRLEN);
      fprintf(stderr,"Destination passed in %s and response %s\n", \
                     temp_addr_name, temp_addr_name1);

   }//if (TROUBLESHOOT==8) {

   while ((bottom <= top) && (result != ZERO)) {

      middle = (bottom + top) / 2;
      result = cmpaddr(&response, &(alias_list[middle].original_address),
                       128);
      if (TROUBLESHOOT==8) {

        fprintf(stderr, "The value of bottom = %i\n", bottom);
        fprintf(stderr, "The value of top    = %i\n", top);
        fprintf(stderr, "The value of middle = %i\n", middle);
        fprintf(stderr, "The value of result = %i\n", result);

      }//if (TROUBLESHOOT==8) {
```

```
     if (result == ZERO){

        /*****************************************************************
        *
        * If we are here we found the node
        *
        *****************************************************************/

        if (cmpaddr(&alias_list[middle].resolved_address, &empty, 128) ==
            ZERO) {

           alias_list[middle].resolved_address = destination;

           if (TROUBLESHOOT==8) {

              char  temp_addr_name[INET6_ADDRSTRLEN+1];
              char  temp_addr_name1[INET6_ADDRSTRLEN+1];
              inet_ntop(AF_INET6, &alias_list[middle].original_address,
                        temp_addr_name, INET6_ADDRSTRLEN);
              inet_ntop(AF_INET6, &destination , temp_addr_name1,
                        INET6_ADDRSTRLEN);
              fprintf(stderr,"Insert for org addr %s the addr %s\n", \
                             temp_addr_name, temp_addr_name1);

           }//if (TROUBLESHOOT==8) {

        } else {

           snprintf(textbuffer, MAX_MESSAGE,"ALIAS INSERT: WARNING: Tried"\
                                    " to insert address already on list");
           call_sys_log(textbuffer);

        }//if (cmpaddr(&alias_list[middle].resolved_address, &empty, 128) ==

     } else if (result < ZERO) {

        top = middle - 1;

     } else {

        bottom = middle + 1;

     }//if (result == ZERO){

   }//while (bottom <= top) {

}//void insert_on_node_list(struct in6_addr source_address, struct in6_addr
```

ALIAS_MAIN_HELPER.H

```
/*****************************************************************************
 * This is the header file for all the programs that help alias run
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: alias_main_helper.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_ALIAS_MAIN_HELPER
#define INCLUSION_GUARD_PROGRAM_ALIAS_MAIN_HELPER

   // Processes all the received packets
   int  process_alias_receive(struct processing_source processing_list[]);
   // Fill the process list
   void fill_alias_list(struct processing_source processing_list[]);
   // Generates new probes
   void generate_alias_probes(struct processing_source processing_list[]);
   // Checks to make sure we are not done
   int  alias_done_check(struct processing_source processing_list[]);
   // Updates the edge file
   int  alias_update();
   // Saves the results
   int  alias_save();

#endif //INCLUSION_GUARD_PROGRAM_ALIAS_MAIN_HELPER
```

ALIAS_OFFLINE.C

```
/*****************************************************************************
* This is the main program of the alias offline resolver and controls
* all operation
*
*
* Author: Robert J. Poulin, Capt, USAF
*
* Program name: alias_offline.c
*
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <netinet/in.h>
#include <libnet.h>
#include <signal.h>
#include <unistd.h>
#include <wait.h>
#include "probe_main.h"
#include "probe_generator.h"
#include "probe_receive.h"
#include "probe_utils.h"
#include "probe_loader.h"
#include "probe_ckpnt.h"
#include "alias_main_helper.h"
#include "alias_offline.h"
#include "sys_log.h"
#include <sys/socket.h>
#include <arpa/inet.h>
#include <pcap.h>


#define _GNU_SOURCE
#define TROUBLESHOOT 1

/*****************************************************************************
*
* Define constants
*
*****************************************************************************/

   const  int               IPC_ERROR        = -1; //The msg func calls -1 err
   const  int               LIBNET_ERROR     = -1; //Libnet calls -1 an error
   const  int               positive         =  1; //Flg positive direction
   const  int               negative         = -1; //Flg negative direction


/*****************************************************************************
*
* Define externals and globals
*
*****************************************************************************/

   extern int errno;      // Set by calls to the library functions

   struct generator_buffer  *message_to_generator;  //Buffer for generator msg
   struct receive_buffer    *message_from_receiver; //Buffer for receive msg
   struct sources           *source_list;           //Pointer to list of srcs
   struct edges             *edge_list;             //Pointer to list of edges
   struct global_stops      *global_stop_list;      //Pntr to global stop set
          void*             *probe_source_stop_list;//Pointer to local stop set
   struct alias_nodes       *alias_list;            //Pointer to alias list

   struct messagebuffer     sendbuffer, recvbuffer;
```

```
   struct in6_addr          empty;                   //Flag for no address
   struct in6_addr          anonymous_address;       //Addr for anonymous nodes

          int               queueid;                          //Queue id nmb

          char          our_ipv6_addr_name[INET6_ADDRSTRLEN+1];//Our IPv6 addr
          char          tunnelip[INET_ADDRSTRLEN+1];         //Tunnel IPv4 addr
          int           main_done         = PLZCONTINUE;     //Loop stop flg

   int    number_of_sources        = ZERO;   // The number of src for probes
   int    number_of_edges          = ZERO;   // The number of edges found
   int    number_of_global_stops   = ZERO;   // The number of global stops
   int    number_of_nodes          = ZERO;   // The number of total nodes
   int    probe_generator_pid      = ZERO;   // Probe generator PID
   int    sys_log_pid              = ZERO;   // Sys log PID
   int    probe_receive_pid        = ZERO;   // Probe receiver PID
   int    is_main                  = TRUE;   // Tells us if we are main
   time_t check_point_time         = ZERO;   // This records checkpoint time
   unsigned int  anonymous_number  = ZERO;   // The number for anonymous nodes
   int    NUMBER_OF_PACKETS        = ZERO;   // The number of pkts per probe
   int    MIN_RESPONSE             = ZERO;   // Minimum # of response for valid
   int    MAX_ANONYMOUS            = ZERO;   // Needed for probe_main.h
   double DEFAULT_P_VALUE          = 0.0;    // Needed for probe_main.h
   int    WAIT_TIME                = ZERO;   // the time to wait for response
   int    CHECK_POINT_TIME         = ZERO;   // Needed for probe_main.h
   int    deleted_edges            = ZERO;   // Number of edges deleted

/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/

 void main_handler(int sig) {

    int pid     = ZERO;
    int status  = ZERO;

    if (is_main == TRUE) {

       fprintf(stderr, "ALIAS system received the message and is stopping," \
                       " Please wait!\n");
       main_done = STOP;

    }//if (is_main == TRUE) {


}// void main_handler(int sig) {

 void child_handler(int sig) {

    char textbuffer[MAX_MESSAGE];  // Buffer to hold our message
    int  status;
    int  pid         = ZERO;


    pid = wait(&status);
    if (pid == sys_log_pid) {

       fprintf(stderr,"ALIAS MAIN: Sys log stopped, if this is unplanned " \
                      "check log\n");
       fprintf(stderr,"ALIAS MAIN: The sys log process stopped with status: " \
                      "%i\n", status);
       sys_log_pid = ZERO;
       if (TROUBLESHOOT != 99) {

          main_done   = STOP;     // If sys-log stopped we don't record so stop

       }//if (TROUBLESHOOT != 99) {
```

```
    } else if (pid == probe_generator_pid) {

       fprintf(stderr,"ALIAS MAIN: Probe Generator stopped, if this is " \
                      "unplanned check log\n");
       snprintf(textbuffer, MAX_MESSAGE, "ALIAS: The probe generator stopped" \
                                         " with status: %i", status);
       call_sys_log(textbuffer);
       probe_generator_pid = ZERO;
       if (TROUBLESHOOT != 99) {

          main_done = STOP;       // If generator stopped no reason to continue

       }//if (TROUBLESHOOT != 99) {

    } else if (pid == probe_receive_pid) {

       fprintf(stderr,"ALIAS MAIN: Probe Receiver stopped, if this is " \
                      "unplanned check log\n");
       snprintf(textbuffer, MAX_MESSAGE, "ALIAS MAIN: The probe receiver " \
                                         "stopped with status: %i", status);
       call_sys_log(textbuffer);
       probe_receive_pid = ZERO;
       if (TROUBLESHOOT != 99) {

          main_done = STOP;     // If receiver stopped no reason to continue

       }//if (TROUBLESHOOT != 99) {

    } else {

       snprintf(textbuffer, MAX_MESSAGE, "ALIAS MAIN: Received a signal but " \
                                         "pid did not match any running " \
                                         "process, status = : %i\n", status);
       call_sys_log(textbuffer);

    }//if (pid == sys_log_pid) {

}//void child_handler(int sig) {

/*****************************************************************************
 *
 * Main function
 *
 *****************************************************************************/

int main(int argc, char *argv[]){

   /*****************************************************************************
   *
   * Define variables needed by main
   *
   *****************************************************************************/

   struct                   msqid_ds msqid_ds, *buf; //May need this for msg
                                                     //system to check status
                            buf           = &msqid_ds;
   int                      continue_flag = ZERO;    // Flag to tell if any of
                                                     // the processes should
                                                     // continue or quit
   struct in6_addr          destination;             // Used by main to store
                                                     // destination addresses
   int                      fork_error    = FALSE;   // Flag to tell us if we
                                                     // had an error forking
   int                      forked_pid    = ZERO;    // PID returned by fork
   int                      hop_limit     = ZERO;    // Hop limit used by main
   key_t                    key           = KEY;     // This will be the value
                                                     // used to determine
                                                     // our queue id
```

```
libnet_t                *packet_handle;          // Initial handle for pckt
int                     payload_type  = ZERO;    // Used by main to store
                                                 // payload type, ICMP or
                                                 // UDP
int                     rcvsuccess    = ZERO;    // value of our receive op
int                     sendsuccess   = ZERO;    // value of our send op
struct in6_addr         source;                  // Used by main to store
                                                 // source addresses
int                     source_port   = ZERO;    // Used by main to store
                                                 // the source port number
char                    textbuffer[MAX_MESSAGE]; // Buffer to hold our
                                                 // messages for sys-log
int                     count         = ZERO;    // General cntr for loops
int                     seconds       = ZERO;    // Seed for random number
int                     processing_done = FALSE; // Ensures we don't save
                                                 // unless we are done

/*****************************************************************************
*
* Set up initial values
*
*****************************************************************************/

time(&seconds); // Get value from system clock and place in seconds for seed
srand((unsigned int) seconds);  //Convert seconds to a unsigned int and seed

int test = inet_pton(AF_INET6, "fe80::0", &empty);
if (test <= ZERO){

   fprintf(stderr,"ALIAS MAIN: Error unable to initialize values " \
                  "\"empty\"\n");
   return -1;

}//if (test <= ZERO){

 /*****************************************************************************
 *
 * Set up the message queue for all to use
 *
 *****************************************************************************/

 queueid = msgget(key, 0766 | IPC_CREAT);    // Get our message queue
 if (queueid == IPC_ERROR) {

   fprintf(stderr,"ALIAS MAIN: ERROR: Message Get Failed!!!!\n");
   print_IPC_error("ALIAS MAIN");
   return -1;

} else {

   printf("ALIAS MAIN:Message get succeeded, Queue =  %i\n", queueid );

}//if (queueid == IPC_ERROR) {

/*****************************************************************************
*
* Kick off all the other functions
*
*****************************************************************************/

is_main    = TRUE;
fork_error = FALSE;

/*****************************************************************************
*
* Kick off syslog
*
*****************************************************************************/
forked_pid = fork();
```

```
if (forked_pid == ZERO) {

   //We are the child
   is_main = FALSE;
   sys_log(); //We forked successfully we are the child!!!

} else if (forked_pid < ZERO) {

   fprintf(stderr,"ALIAS MAIN: ERROR: trying to fork sys log, return = " \
                  "%i\n", forked_pid);
   fork_error = TRUE;

} else {

   //We are main and everything worked
   sys_log_pid = forked_pid;
   forked_pid  = ZERO;
   if (TROUBLESHOOT == 1){

      fprintf(stderr,"The pid for sys  is: %i\n",sys_log_pid);

   }//if (TROUBLESHOOT == 1){

}//if (forked_pid == ZERO) {

/*****************************************************************************
*
* Setup before we begin probing
*
*****************************************************************************/

if (is_main && fork_error == FALSE) {

   int    startup      = ZERO;        // The success or failure of startup
   int    done         = PLZCONTINUE; // This causes us to loop until we
                                      // are done

   if (sys_log_pid == ZERO){

      fprintf(stderr, "We had a bad startup, shutting down!!!\n");
      startup = 2;

   }//if (sys_log_pid == ZERO)

   if (startup == ZERO) {

      fprintf(stdout,"Beginning the load of alias list\n");
      startup = load_alias_list();
      if (startup != ZERO) {

         fprintf(stderr, "Error loading alias list!\n");

      }//if (startup != ZERO) {

   }//if (startup == ZERO) {


   /*****************************************************************************
   *
   * Kick off the signal handlers
   *
   *****************************************************************************/

   signal(SIGCHLD, child_handler);

   /*****************************************************************************
   *
   * This is the main routine that accomplishes all the tasks
```

```
*
**********************************************************************/

if (TROUBLESHOOT==1) {

   fprintf(stderr,"We are about to start resolving\n");

}//if (TROUBLESHOOT==1) {

if (is_main && (startup == ZERO)) {

   int update = alias_update();
   if (update == ZERO) {

      if(alias_ckpnt() != ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Failure " \
                                          "updating the edges");
         call_sys_log(textbuffer);
         fprintf(stderr, "ALIAS MAIN: We had a failure updating " \
                         "the edges, check the log for error\n");

      }//if(alias_ckpnt() != ZERO) {

   } else {

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Failure saving " \
                                       "the edges");
      call_sys_log(textbuffer);
      fprintf(stderr, "ALIAS MAIN: We had a failure saving the " \
                      "edges, check the log for error\n");

   }//if (update == ZERO) {

} else {

   fprintf(stderr,"ALIAS MAIN: ERROR in loading startup files.  " \
                  "Shutting down!\n");

}//if (startup == ZERO) {

/**********************************************************************
*
* Save if we were successfull
*
**********************************************************************/

if (startup == ZERO) {

   int temp = alias_save();
   if (temp != ZERO) {

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Failure saving");
      call_sys_log(textbuffer);
      fprintf(stderr, "ALIAS MAIN: We had a failure saving, check the " \
                      "log for error\n");

   } else {

      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Saving successful " \
                                       "of nodes succesful");
      call_sys_log(textbuffer);
      fprintf(stderr,"%s\n", textbuffer);
      snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Deleted %i edges", \
                                       deleted_edges);
      call_sys_log(textbuffer);
      fprintf(stderr,"%s\n", textbuffer);
      int unique_nodes = ZERO;
      for (count=ZERO; count < number_of_nodes; count++) {
```

```
         if(cmpaddr(&alias_list[count].original_address,
                    &alias_list[count].resolved_address, 128) == ZERO) {

            unique_nodes++;

         }//if(cmpaddr(&alias_list[count].original_address,

       }//for (count=ZERO; count < number_of_nodes; count++) {
       snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Number of original " \
                                        "nodes is %i", number_of_nodes);
       call_sys_log(textbuffer);
       fprintf(stderr,"%s\n", textbuffer);

       snprintf(textbuffer, MAX_MESSAGE,"ALIAS MAIN: Number of unqiue " \
                                        "nodes left is %i", unique_nodes);
       call_sys_log(textbuffer);
       fprintf(stderr,"%s\n", textbuffer);

    }//if (temp != ZERO) {

 }//if (startup == ZERO) {

 /*************************************************************************
*
* Kill the sys log
*
*************************************************************************/
sendbuffer.type = SYSLOG;
snprintf(sendbuffer.text, 5, "QUIT");

sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                     sizeof(sendbuffer.text), IPC_NOWAIT);
if(sendsuccess != ZERO){

   fprintf(stderr,"ALIAS MAIN: ERROR: Unable to tell SYS LOG to STOP\n");
   print_IPC_error("ALIAS MAIN");

}//if(sendsuccess != ZERO){

/*************************************************************************
*
* Wait for the sys log to stop
*
*************************************************************************/

count = 0;
while((sys_log_pid  != ZERO) && (count <= 5)){

   printf("ALIAS MAIN: Waiting for sys log to finish\n");
   count++;
   sleep(1);

}//while((sys_log_pid  != ZERO) && (count <= 5)){

//Destroy the queue and clean up
int destroy = msgctl(queueid, IPC_RMID, buf);
if(destroy != ZERO){

   fprintf(stderr,"ALIAS MAIN: ERROR: Unable to delete the queue, error" \
                  " number = %i\n", destroy);
   print_IPC_error("ALIAS MAIN");

} else {

   fprintf(stderr,"ALIAS MAIN: Message QUEUE destroyed\n");

}//if(destroy != ZERO){
```

```
   }// if (is_main && fork_error == FALSE) {
}//int main(int argc, char *argv[])
```

ALIAS_OFFLINE.H

```
/*****************************************************************************
 * This is the header file of the main offline alias engine *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: alias_offline.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_ALIAS_OFFLINE
#define INCLUSION_GUARD_PROGRAM_ALIAS_OFFLINE


#endif //INCLUSION_GUARD_PROGRAM_ALIAS_OFFLINE
```

ANON_MAIN.C

```
/*****************************************************************************
* This is the anonymous resolver main engine and controls all operation
*
*
* Author: Robert J. Poulin, Capt, USAF
*
* Program name: anon_main.c
*
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <netinet/in.h>
#include <signal.h>
#include <unistd.h>
#include <wait.h>
#include "probe_main.h"
#include "probe_utils.h"
#include "probe_loader.h"
#include "probe_ckpnt.h"
#include "probe_edges.h"
#include "sys_log.h"
#include <sys/socket.h>
#include <arpa/inet.h>
#include <time.h>



#define _GNU_SOURCE
#define TROUBLESHOOT 0    // 1  - Turn code to troubleshoot routine
                          // 99 - Turn on code to eliminate nodes to no where
#define LEFT         1
#define RIGHT        2


/*****************************************************************************
*
* Define constants
*
*****************************************************************************/

   // The messaging function calls -1 an error
   const  int               IPC_ERROR        = -1;
   // The messaging function calls -1 an error
   const  int               LIBNET_ERROR     = -1;
   // Flag stating we are going in the positive direction
   const  int               positive         =  1;
   // Flag stating we are going in the negative direction
   const  int               negative         = -1;


/*****************************************************************************
*
* Define externals and globals
*
*****************************************************************************/

   // Set by calls to the library functions to define errors encountered
   extern int errno;

   //Buffer for messages to the generator function
   struct generator_buffer  *message_to_generator;
   // Buffer for messages from receive  function
   struct receive_buffer    *message_from_receiver;
```

```
  // Pointer to the list of sources
  struct sources           *source_list;
  // Pointer to the list of edges
  struct edges             *edge_list;
  // Pointer to the global stop set
  struct global_stops      *global_stop_list;
  // Pointer to the local stop set
         void*             *probe_source_stop_list;
  // Pointer to the alias list for alias resolution
  struct alias_nodes       *alias_list;

  struct messagebuffer     sendbuffer, recvbuffer;
  // Flag address stating there is no address
  struct in6_addr          empty;
  // address used to mark anonymous nodes
  struct in6_addr          anonymous_address;

  // The id number of the queue we are using, needs to be global for all
  // the programs will use it
         int               queueid;

  // Our IPv6 address
         char              our_ipv6_addr_name[INET6_ADDRSTRLEN+1];
  // The ipv4 address of the tunnel endpoint
         char              tunnelip[INET_ADDRSTRLEN+1];
  // Loop variable telling us to stop
         int               main_done         = PLZCONTINUE;

  // The total number of sources for our probes
  int    number_of_sources        = ZERO;
  // The total number of edges found
  int    number_of_edges          = ZERO;
  // The number in the global stop table
  int    number_of_global_stops   = ZERO;
  // The number of total nodes in the alias list
  int    number_of_nodes          = ZERO;
  // Used to store the pid of the probe generator
  int    probe_generator_pid      = ZERO;
  // Used to store the pid of the sys log facility
  int    sys_log_pid              = ZERO;
  // Used to store the pid of the probe receiver
  int    probe_receive_pid        = ZERO;
  // Tells us if we are main
  int    is_main                  = TRUE;
  // This records our checkpoint time
  time_t check_point_time         = ZERO;
  // The number we are using in the anonymous nodes
  unsigned int  anonymous_number  = ZERO;
  // The number of packets to be sent for each probe
  int    NUMBER_OF_PACKETS        = ZERO;
  // The minimum number of reesponse we will consider valid
  int    MIN_RESPONSE             = ZERO;
  // The maximum amount of anonymous nodes before we stop probing
  int    MAX_ANONYMOUS            = ZERO;
  // The default value using the cdf function to determine starting hop count
  double DEFAULT_P_VALUE          = 0.0;
  // The time we will wait for a response
  int    WAIT_TIME                = ZERO;
  // How much time should elapse before we check point
  int    CHECK_POINT_TIME         = ZERO;
  // The current time in system format
  time_t    curtime;
  struct tm *loctime;


/*****************************************************************************
 *
 * Subroutines
 *
```

```
 ******************************************************************************/

 void child_handler(int sig) {

    char textbuffer[MAX_MESSAGE];  // Buffer to hold our message
    int  status;
    int  pid         = ZERO;


    pid = wait(&status);
    if (pid == sys_log_pid) {

       fprintf(stderr,"PROBE MAIN: Sys log stopped, if this is unplanned " \
                      "check log\n");
       fprintf(stderr,"PROBE MAIN: The sys log process stopped with " \
                      "status: %i\n", status);
       sys_log_pid = ZERO;

    } else {

       snprintf(textbuffer, MAX_MESSAGE, "We received a signal but pid did " \
                "not match any running process, status = : %i\n", status);
       call_sys_log(textbuffer);

    }//if (pid == sys_log_pid) {

}//void child_handler(int sig) {

/******************************************************************************
 *
 * Main function
 *
 ******************************************************************************/

int main(int argc, char *argv[]){

   /*****************************************************************************
   *
   * Define variables needed by main
   *
   *****************************************************************************/

   // We may need this by the messaging system to check status
   struct                   msqid_ds msqid_ds, *buf;

                            buf           = &msqid_ds;
   // Flag to tell if any of the processes should continue or quit
   int                      continue_flag = ZERO;
   // Used by main to store destination addresses
   struct in6_addr          destination;
   // Flag to tell us if we had an error forking
   int                      fork_error    = FALSE;
   // PID returned by fork command
   int                      forked_pid    = ZERO;
   // Hop limit used by main
   int                      hop_limit     = ZERO;
   // This will be the value used to determine our queue id
   key_t                    key           = KEY;
   // The value of our receive op!!
   int                      rcvsuccess    = ZERO;
   // The value of our send operation!!
   int                      sendsuccess   = ZERO;
   // Buffer to hold our messages for sys-log
   char                     textbuffer[MAX_MESSAGE];
   // General counter for loops
   int                      count         = ZERO;
   // Pointer to the list of edges
   struct edges             *old_edge_list;
   int                      old_number_of_edges = ZERO;
```

```
 /*************************************************************************
 *
 * Set up initial value of empty
 *
 *************************************************************************/

 int test = inet_pton(AF_INET6, "fe80::0", &empty);
 if (test <= ZERO){

    fprintf(stderr,"ANON MAIN: Error unable to initialize " \
                   "values \"empty\"\n");
    return -1;

 }//if (test <= ZERO){

 /*************************************************************************
 *
 * Set up the message queue for all to use
 *
 *************************************************************************/

 queueid = msgget(key, 0766 | IPC_CREAT);    // Get our message queue
 if (queueid == IPC_ERROR) {

   fprintf(stderr,"ANON MAIN: ERROR: Message Get Failed!!!!\n");
   print_IPC_error("ANON MAIN");
   return -1;

} else {

   printf("ANON MAIN:Message get succeeded, Queue =  %i\n", queueid );

}//if (queueid == IPC_ERROR) {

/*************************************************************************
*
* Kick off all the other functions
*
*************************************************************************/

is_main    = TRUE;
fork_error = FALSE;

/*************************************************************************
*
* Kick off syslog
*
*************************************************************************/
forked_pid = fork();

if (forked_pid == ZERO) {

   //We are the child
   is_main = FALSE;
   sys_log(); //We forked successfully we are the child!!!

} else if (forked_pid < ZERO) {

   fprintf(stderr,"ANON MAIN: ERROR: trying to fork sys log, return =" \
                  " %i\n", forked_pid);
   fork_error = TRUE;

} else {

   //We are main and everything worked
   sys_log_pid = forked_pid;
   forked_pid  = ZERO;
```

```
   if (TROUBLESHOOT == 1){

      fprintf(stderr,"The pid for sys log is: %i\n",sys_log_pid);

   }//if (TROUBLESHOOT == 1){

}//if (forked_pid == ZERO) {

/*****************************************************************************
*
* Kick off main process
*
*****************************************************************************/

if (is_main && fork_error == FALSE) {

   // The success or failure of startup
   int    startup      = ZERO;
   // This causes us to loop until we are done
   int    done         = PLZCONTINUE;

   if (sys_log_pid == ZERO){

      fprintf(stderr, "We had a bad startup, shutting down!!!\n");
      startup = 2;

   }//if ((sys_log_pid == ZERO) || (probe_generator_pid == ZERO) || (probe_r

   if (startup == ZERO) {

      startup = load_edges();
      if (startup != ZERO) {

         fprintf(stderr, "Error loading edges!\n");

      }//if (startup != ZERO) {

   }//if (startup == ZERO) {

   /*****************************************************************************
   *
   * Kick off the signal handlers
   *
   *****************************************************************************/

   signal(SIGCHLD, child_handler);

   /*****************************************************************************
   *
   * Start resolving
   *
   *****************************************************************************/

   printf("Starting the anonymous resolution\n");

   if (startup == ZERO) {

     old_edge_list = edge_list;
     edge_list = (struct edges*) malloc(MAX_EDGES * sizeof(struct edges));
     old_number_of_edges = number_of_edges;
     number_of_edges = ZERO;
     int current_edge = ZERO;

     /*****************************************************************************
     *
     * Locate each anonymous node and reduce
     *
     *****************************************************************************/
     int a_number = ZERO;
```

```
for(a_number=ZERO;a_number<anonymous_number;a_number++) {

   anonymous_address.in6_u.u6_addr32[0] = htonl(0xfe800000);
   anonymous_address.in6_u.u6_addr32[1] = 0x00000000;
   anonymous_address.in6_u.u6_addr32[2] = 0x00000000;
   anonymous_address.in6_u.u6_addr32[3] = htonl(a_number);

   struct in6_addr left   = empty;
   struct in6_addr right  = empty;
   struct in6_addr middle = empty;
   int    found           = FALSE;
   int    displayed       = FALSE;


   for(count=0;count<old_number_of_edges;count++) {

      /*************************************************************
      *
      * Find the left and right pair
      *
      *************************************************************/

      if ((cmpaddr(&old_edge_list[count].edge_v1,
                   &empty, 16) == ZERO) ||
           (cmpaddr(&old_edge_list[count].edge_v2,
                   &empty, 16) == ZERO)) {

         if (cmpaddr(&left, &empty, 16) == ZERO) {

            if (cmpaddr(&old_edge_list[count].edge_v1,
                        &anonymous_address, 128) == ZERO) {

               if (cmpaddr(&old_edge_list[count].edge_v2,
                           &empty, 16) != ZERO){

                  left   = old_edge_list[count].edge_v2;
                  middle = anonymous_address;

               }//if (cmpaddr(&old_edge_list[count].edge_v2, &empty, 16

            } else if (cmpaddr(&old_edge_list[count].edge_v2,
                               &anonymous_address, 128) == ZERO) {

               if (cmpaddr(&old_edge_list[count].edge_v1,
                           &empty, 16) != ZERO){

                  left   = old_edge_list[count].edge_v1;
                  middle = anonymous_address;

               }//if (cmpaddr(&old_edge_list[count].edge_v1, &empty, 16

            }//if (cmpaddr(&old_edge_list[count].edge_v1, &anonymous_ad

         } else {

            if (cmpaddr(&old_edge_list[count].edge_v1,
                        &anonymous_address, 128) == ZERO) {

               if (cmpaddr(&old_edge_list[count].edge_v2,
                           &empty, 16) != ZERO){

                  right   = old_edge_list[count].edge_v2;
                  found   = TRUE;

               }//if (cmpaddr(&old_edge_list[count].edge_v2, &empty, 16

            } else if (cmpaddr(&old_edge_list[count].edge_v2,
                               &anonymous_address, 128) == ZERO) {
```

```
            if (cmpaddr(&old_edge_list[count].edge_v1,
                        &empty, 16) != ZERO){

               right   = old_edge_list[count].edge_v1;
               found   = TRUE;

            }//if (cmpaddr(&old_edge_list[count].edge_v1, &empty, 16

         }//if (cmpaddr(&old_edge_list[count].edge_v1, &anonymous_ad

         if ((TROUBLESHOOT==1) && (found == TRUE) &&
             (displayed == FALSE)) {

            char  temp_addr_namel[INET6_ADDRSTRLEN+1];
            char  temp_addr_namem[INET6_ADDRSTRLEN+1];
            char  temp_addr_namer[INET6_ADDRSTRLEN+1];
            inet_ntop(AF_INET6, &left , temp_addr_namel,
                      INET6_ADDRSTRLEN);
            inet_ntop(AF_INET6, &middle , temp_addr_namem,
                      INET6_ADDRSTRLEN);
            inet_ntop(AF_INET6, &right , temp_addr_namer,
                      INET6_ADDRSTRLEN);
            snprintf(textbuffer, MAX_MESSAGE,"ANON MAIN: Found " \
                     "Right = %s and middle = %s Left = %s", \
                      temp_addr_namer, temp_addr_namem, \
                      temp_addr_namel);
            call_sys_log(textbuffer);
            fprintf(stderr,"%s\n", textbuffer);
            displayed = TRUE;

         }//if (TROUBLESHOOT==1) {

      }//if (cmpaddr(&left, &empty, 128) == ZERO) {

   }//if ((cmpaddr(&old_edge_list[count].edge_v1, &empty, 16) == ZER

}//for(count=0;count<number_of_edges;count++) {

if (found == TRUE) {

   char  temp_addr_namel[INET6_ADDRSTRLEN+1];
   char  temp_addr_namem[INET6_ADDRSTRLEN+1];
   char  temp_addr_namer[INET6_ADDRSTRLEN+1];
   inet_ntop(AF_INET6, &left , temp_addr_namel, INET6_ADDRSTRLEN);
   inet_ntop(AF_INET6, &middle , temp_addr_namem, INET6_ADDRSTRLEN);
   inet_ntop(AF_INET6, &right , temp_addr_namer, INET6_ADDRSTRLEN);
   snprintf(textbuffer, MAX_MESSAGE,"ANON MAIN: looking for three" \
                                    " some: ");
   call_sys_log(textbuffer);
   if (TROUBLESHOOT==1) {

      fprintf(stderr,"%s\n", textbuffer);

   }//if (TROUBLESHOOT==1) {
   snprintf(textbuffer, MAX_MESSAGE,"ANON MAIN: Left = : %s", \
                                     temp_addr_namel);
   call_sys_log(textbuffer);
   if (TROUBLESHOOT==1) {

      fprintf(stderr,"%s\n", textbuffer);

   }//if (TROUBLESHOOT==1) {
   snprintf(textbuffer, MAX_MESSAGE,"ANON MAIN: Middle = : %s", \
                                    temp_addr_namem);
   call_sys_log(textbuffer);
   if (TROUBLESHOOT==1) {

      fprintf(stderr,"%s\n", textbuffer);
```

```
}//if (TROUBLESHOOT==1) {
snprintf(textbuffer, MAX_MESSAGE,"ANON MAIN: Right = : %s", \
                                temp_addr_namer);
call_sys_log(textbuffer);
if (TROUBLESHOOT==1) {

   fprintf(stderr,"%s\n", textbuffer);

}//if (TROUBLESHOOT==1) {

struct in6_addr locate;
int             we_need_to_match = ZERO;

for(count=ZERO;count<old_number_of_edges;count++) {

   if ((cmpaddr(&old_edge_list[count].edge_v1,
               &empty, 16) == ZERO) ||
      (cmpaddr(&old_edge_list[count].edge_v2,
               &empty, 16) == ZERO)) {

      if (cmpaddr(&old_edge_list[count].edge_v1,
                  &empty, 16) == ZERO) {

         if(cmpaddr(&old_edge_list[count].edge_v1,
                    &anonymous_address, 128) > ZERO) {

            if ((cmpaddr(&old_edge_list[count].edge_v2,
                       &left, 128) == ZERO) ||
               (cmpaddr(&old_edge_list[count].edge_v2,
                        &right, 128) == ZERO)) {

               if (cmpaddr(&old_edge_list[count].edge_v2,
                           &left, 128) == ZERO) {

                   locate = old_edge_list[count].edge_v1;
                   we_need_to_match  = RIGHT;

                   if (TROUBLESHOOT==1) {

                      char  temp_addr_namer[INET6_ADDRSTRLEN+1];
                      inet_ntop(AF_INET6, &locate ,
                              temp_addr_namel, INET6_ADDRSTRLEN);
                      snprintf(textbuffer, MAX_MESSAGE,"ANON "\
                               "MAIN: Found the left and need"\
                               " to locate %s", temp_addr_namel);
                      call_sys_log(textbuffer);
                      fprintf(stderr,"%s\n", textbuffer);

                   }//if (TROUBLESHOOT==1) {

               } else {

                   locate = old_edge_list[count].edge_v1;
                   we_need_to_match  = LEFT;

                   if (TROUBLESHOOT==1) {

                      char  temp_addr_namer[INET6_ADDRSTRLEN+1];
                      inet_ntop(AF_INET6, &locate ,
                              temp_addr_namel, INET6_ADDRSTRLEN);
                      snprintf(textbuffer, MAX_MESSAGE,"ANON " \
                               "MAIN: Found the right and need" \
                               " to locate %s", temp_addr_namel);
                      call_sys_log(textbuffer);
                      fprintf(stderr,"%s\n", textbuffer);

                   }//if (TROUBLESHOOT==1) {

               }//if (cmpaddr(&old_edge_list[count].edge_v2, &lef
```

```
int local_counter = ZERO;
for(local_counter=ZERO;
    local_counter<old_number_of_edges;
    local_counter++) {

   if((cmpaddr(&locate,
       &old_edge_list[local_counter].edge_v1,
       128)==ZERO) ||
      (cmpaddr(&locate,
       &old_edge_list[local_counter].edge_v2,
       128)==ZERO)) {

      if(we_need_to_match == RIGHT) {

         if((cmpaddr(&right,
           &old_edge_list[local_counter].edge_v1,
           128)==ZERO) ||
            (cmpaddr(&right,
           &old_edge_list[local_counter].edge_v2,
           128)==ZERO)) {

            if(cmpaddr(&right,
           &old_edge_list[local_counter].edge_v1,
           128)==ZERO) {

     char  temp_addr_nameold[INET6_ADDRSTRLEN+1];
     char  temp_addr_namem[INET6_ADDRSTRLEN+1];
     inet_ntop(AF_INET6,
         &old_edge_list[local_counter].edge_v2,
         temp_addr_nameold, INET6_ADDRSTRLEN);
     inet_ntop(AF_INET6, &middle ,
         temp_addr_namem, INET6_ADDRSTRLEN);
     snprintf(textbuffer, MAX_MESSAGE,"ANON " \
              "MAIN: Replacing: %s with %s", \
              temp_addr_nameold, temp_addr_namem);
               call_sys_log(textbuffer);
               if (TROUBLESHOOT==1) {

               fprintf(stderr,"%s\n", textbuffer);

               }//if (TROUBLESHOOT==1) {

    old_edge_list[local_counter].edge_v2 = middle;
    if(cmpaddr(&old_edge_list[count].edge_v1,
               &empty, 16)==ZERO) {

        old_edge_list[count].edge_v1 = middle;

    } else {

        old_edge_list[count].edge_v2 = middle;

    }//if(cmpaddr(&old_edge_list[count].edge_v1, &

            } else {

     char  temp_addr_nameold[INET6_ADDRSTRLEN+1];
     char  temp_addr_namem[INET6_ADDRSTRLEN+1];
     inet_ntop(AF_INET6,
         &old_edge_list[local_counter].edge_v1,
         temp_addr_nameold, INET6_ADDRSTRLEN);
     inet_ntop(AF_INET6, &middle,
               temp_addr_namem, INET6_ADDRSTRLEN);
     snprintf(textbuffer, MAX_MESSAGE,"ANON "\
              "MAIN: Replacing: %s with %s", \
              temp_addr_nameold, temp_addr_namem);
               call_sys_log(textbuffer);
               if (TROUBLESHOOT==1) {
```

```
          fprintf(stderr,"%s\n", textbuffer);

          }//if (TROUBLESHOOT==1) {

old_edge_list[local_counter].edge_v1 = middle;
if(cmpaddr(&old_edge_list[count].edge_v1,
           &empty, 16)==ZERO) {

   old_edge_list[count].edge_v1 = middle;

} else {

   old_edge_list[count].edge_v2 = middle;

}//if(cmpaddr(&old_edge_list[count].edge_v1,

       }//if(cmpaddr(&right,&old_edge_list[lo

    }//if((cmpaddr(&right,&old_edge_list[loca

 } else {

if((cmpaddr(&left,
    &old_edge_list[local_counter].edge_v1,
    128)==ZERO) ||
   (cmpaddr(&left,
    &old_edge_list[local_counter].edge_v2,
    128)==ZERO)) {

if(cmpaddr(&left,
   &old_edge_list[local_counter].edge_v1,
   128)==ZERO) {

char  temp_addr_nameold[INET6_ADDRSTRLEN+1];
char  temp_addr_namem[INET6_ADDRSTRLEN+1];
inet_ntop(AF_INET6,
    &old_edge_list[local_counter].edge_v2,
    temp_addr_nameold, INET6_ADDRSTRLEN);
inet_ntop(AF_INET6, &middle,
          temp_addr_namem, INET6_ADDRSTRLEN);
snprintf(textbuffer, MAX_MESSAGE,"ANON " \
         "MAIN: Replacing: %s with %s", \
         temp_addr_nameold, temp_addr_namem);
          call_sys_log(textbuffer);
          if (TROUBLESHOOT==1) {

          fprintf(stderr,"%s\n", textbuffer);

          }//if (TROUBLESHOOT==1) {

old_edge_list[local_counter].edge_v2 = middle;
if(cmpaddr(&old_edge_list[count].edge_v1,
           &empty, 16)==ZERO) {

   old_edge_list[count].edge_v1 = middle;

} else {

   old_edge_list[count].edge_v2 = middle;

}//if(cmpaddr(&old_edge_list[count].edge_v1,

       } else {

char  temp_addr_nameold[INET6_ADDRSTRLEN+1];
char  temp_addr_namem[INET6_ADDRSTRLEN+1];
inet_ntop(AF_INET6,
    &old_edge_list[local_counter].edge_v1,
```

```
                    temp_addr_nameold, INET6_ADDRSTRLEN);
                inet_ntop(AF_INET6, &middle,
                    temp_addr_namem, INET6_ADDRSTRLEN);
                snprintf(textbuffer, MAX_MESSAGE,"ANON " \
                         "MAIN: Replacing: %s with %s", \
                         temp_addr_nameold, temp_addr_namem);
                call_sys_log(textbuffer);
                          if (TROUBLESHOOT==1) {

                          fprintf(stderr,"%s\n", textbuffer);

                          }//if (TROUBLESHOOT==1) {

                old_edge_list[local_counter].edge_v1 = middle;
                if(cmpaddr(&old_edge_list[count].edge_v1,
                           &empty, 16)==ZERO) {

                  old_edge_list[count].edge_v1 = middle;

                } else {

                  old_edge_list[count].edge_v2 = middle;

                }//if(cmpaddr(&old_edge_list[count].edge_v1,

                       }//if(cmpaddr(&right,&old_edge_list[lo

                    }//if((cmpaddr(&left,&old_edge_list[local

                 }//if(we_need_to_match == RIGHT) {

              }//if((cmpaddr(&locate,&old_edge_list[local_cou

           }//for(local_counter=ZERO;local_counter<old_number

        }//if ((cmpaddr(&old_edge_list[count].edge_v2, &left,

     }//if(cmpaddr(&old_edge_list[count].edge_v1, &anonymous_

  } else {

     if(cmpaddr(&old_edge_list[count].edge_v2,
                &anonymous_address, 128) > ZERO) {

        if ((cmpaddr(&old_edge_list[count].edge_v1,
                    &left, 128) == ZERO) ||
           (cmpaddr(&old_edge_list[count].edge_v1,
                    &right, 128) == ZERO)) {

           if (cmpaddr(&old_edge_list[count].edge_v1,
                      &left, 128) == ZERO) {

               locate = old_edge_list[count].edge_v2;
               we_need_to_match  = RIGHT;

               if (TROUBLESHOOT==1) {

                  char  temp_addr_namer[INET6_ADDRSTRLEN+1];
                  inet_ntop(AF_INET6, &locate ,
                            temp_addr_namel, INET6_ADDRSTRLEN);
                  snprintf(textbuffer, MAX_MESSAGE,"ANON " \
                           "MAIN: Found the left and need " \
                           "to locate %s", temp_addr_namel);
                  call_sys_log(textbuffer);
                  fprintf(stderr,"%s\n", textbuffer);

               }//if (TROUBLESHOOT==1) {

           } else {
```

```
    locate = old_edge_list[count].edge_v2;
    we_need_to_match  = LEFT;

    if (TROUBLESHOOT==1) {

       char  temp_addr_namer[INET6_ADDRSTRLEN+1];
       inet_ntop(AF_INET6, &locate ,
                 temp_addr_namel, INET6_ADDRSTRLEN);
       snprintf(textbuffer, MAX_MESSAGE,"ANON " \
                "MAIN: Found the right and need " \
                "to locate %s", temp_addr_namel);
       call_sys_log(textbuffer);
       fprintf(stderr,"%s\n", textbuffer);

    }//if (TROUBLESHOOT==1) {


}//if (cmpaddr(&old_edge_list[count].edge_v2, &lef

int local_counter = ZERO;
for(local_counter=ZERO;
    local_counter<old_number_of_edges;
    local_counter++) {

   if((cmpaddr(&locate,
       &old_edge_list[local_counter].edge_v1,
       128)==ZERO) ||
       (cmpaddr(&locate,
        &old_edge_list[local_counter].edge_v2,
       128)==ZERO)) {

      if(we_need_to_match == RIGHT) {

         if((cmpaddr(&right,
            &old_edge_list[local_counter].edge_v1,
            128)==ZERO) ||
            (cmpaddr(&right,
            &old_edge_list[local_counter].edge_v2,
            128)==ZERO)) {

            if(cmpaddr(&right,
            &old_edge_list[local_counter].edge_v1,
            128)==ZERO) {

     char  temp_addr_nameold[INET6_ADDRSTRLEN+1];
     char  temp_addr_namem[INET6_ADDRSTRLEN+1];
     inet_ntop(AF_INET6,
         &old_edge_list[local_counter].edge_v2,
         temp_addr_nameold, INET6_ADDRSTRLEN);
     inet_ntop(AF_INET6, &middle,
         temp_addr_namem, INET6_ADDRSTRLEN);
     snprintf(textbuffer, MAX_MESSAGE,"ANON " \
              "MAIN: Replacing: %s with %s", \
              temp_addr_nameold, temp_addr_namem);
     call_sys_log(textbuffer);
               if (TROUBLESHOOT==1) {

               fprintf(stderr,"%s\n", textbuffer);

               }//if (TROUBLESHOOT==1) {

     old_edge_list[local_counter].edge_v2 = middle;
     if(cmpaddr(&old_edge_list[count].edge_v1,
                &empty, 16)==ZERO) {

        old_edge_list[count].edge_v1 = middle;

     } else {
```

```
   old_edge_list[count].edge_v2 = middle;

}//if(cmpaddr(&old_edge_list[count].edge_v1,


       } else {

char  temp_addr_nameold[INET6_ADDRSTRLEN+1];
char  temp_addr_namem[INET6_ADDRSTRLEN+1];
inet_ntop(AF_INET6,
    &old_edge_list[local_counter].edge_v1,
    temp_addr_nameold, INET6_ADDRSTRLEN);
inet_ntop(AF_INET6, &middle,
    temp_addr_namem, INET6_ADDRSTRLEN);
snprintf(textbuffer, MAX_MESSAGE,"ANON " \
         "MAIN: Replacing: %s with %s", \
         temp_addr_nameold, temp_addr_namem);
call_sys_log(textbuffer);
          if (TROUBLESHOOT==1) {

          fprintf(stderr,"%s\n", textbuffer);

          }//if (TROUBLESHOOT==1) {

old_edge_list[local_counter].edge_v1 = middle;
if(cmpaddr(&old_edge_list[count].edge_v1,
           &empty, 16)==ZERO) {

   old_edge_list[count].edge_v1 = middle;

} else {

   old_edge_list[count].edge_v2 = middle;

}//if(cmpaddr(&old_edge_list[count].edge_v1,

       }//if(cmpaddr(&right,&old_edge_list[lo

    }//if((cmpaddr(&right,&old_edge_list[loca

 } else {

    if((cmpaddr(&left,
    &old_edge_list[local_counter].edge_v1,
    128)==ZERO) ||
      (cmpaddr(&left,
    &old_edge_list[local_counter].edge_v2,
    128)==ZERO)) {

       if(cmpaddr(&left,
       &old_edge_list[local_counter].edge_v1,
       128)==ZERO) {

char  temp_addr_nameold[INET6_ADDRSTRLEN+1];
char  temp_addr_namem[INET6_ADDRSTRLEN+1];
inet_ntop(AF_INET6,
    &old_edge_list[local_counter].edge_v2,
    temp_addr_nameold, INET6_ADDRSTRLEN);
inet_ntop(AF_INET6, &middle,
    temp_addr_namem, INET6_ADDRSTRLEN);
snprintf(textbuffer, MAX_MESSAGE,"ANON " \
         "MAIN: Replacing: %s with %s", \
         temp_addr_nameold, temp_addr_namem);
call_sys_log(textbuffer);
          if (TROUBLESHOOT==1) {

          fprintf(stderr,"%s\n", textbuffer);
```

```
                                    }//if (TROUBLESHOOT==1) {

                                old_edge_list[local_counter].edge_v2 = middle;
                                if(cmpaddr(&old_edge_list[count].edge_v1,
                                           &empty, 16)==ZERO) {

                                   old_edge_list[count].edge_v1 = middle;

                                } else {

                                   old_edge_list[count].edge_v2 = middle;

                                }//if(cmpaddr(&old_edge_list[count].edge_v1,

                                       } else {

                                char  temp_addr_nameold[INET6_ADDRSTRLEN+1];
                                char  temp_addr_namem[INET6_ADDRSTRLEN+1];
                                inet_ntop(AF_INET6,
                                    &old_edge_list[local_counter].edge_v1,
                                    temp_addr_nameold, INET6_ADDRSTRLEN);
                                inet_ntop(AF_INET6, &middle,
                                    temp_addr_namem, INET6_ADDRSTRLEN);
                                snprintf(textbuffer, MAX_MESSAGE,"ANON " \
                                         "MAIN: Replacing: %s with %s", \
                                         temp_addr_nameold, temp_addr_namem);
                                call_sys_log(textbuffer);
                                          if (TROUBLESHOOT==1) {

                                          fprintf(stderr,"%s\n", textbuffer);

                                          }//if (TROUBLESHOOT==1) {
                                old_edge_list[local_counter].edge_v1 = middle;
                                if(cmpaddr(&old_edge_list[count].edge_v1,
                                           &empty, 16)==ZERO) {

                                   old_edge_list[count].edge_v1 = middle;

                                } else {

                                   old_edge_list[count].edge_v2 = middle;

                                }//if(cmpaddr(&old_edge_list[count].edge_v1,

                                       }//if(cmpaddr(&right,&old_edge_list[lo

                                    }//if((cmpaddr(&left,&old_edge_list[local

                                 }//if(we_need_to_match == RIGHT) {

                              }//if((cmpaddr(&locate,&old_edge_list[local_cou

                           }//for(local_counter=ZERO;local_counter<old_number

                        }//if ((cmpaddr(&old_edge_list[count].edge_v1, &left,

                     }//if(cmpaddr(&old_edge_list[count].edge_v2, &anonymous_

                  }//if (cmpaddr(&old_edge_list[count].edge_v1, &empty, 16) =

               }//if ((cmpaddr(&old_edge_list[count].edge_v1, &empty, 16) ==

            }//for(count=ZERO;count<old_number_of_edges;count++) {

         }//if (found == TRUE) {

      }//for(a_number=ZERO;a_number<anonymous_number;a_number++) {

      /*****************************************************************
```

```
*
*  Reduce the anonymous edges dropped at the source
*
*********************************************************************/

for(count=ZERO; count<anonymous_number;count++) {

   anonymous_address.in6_u.u6_addr32[0] = htonl(0xfe800000);
   anonymous_address.in6_u.u6_addr32[1] = 0x00000000;
   anonymous_address.in6_u.u6_addr32[2] = 0x00000000;
   anonymous_address.in6_u.u6_addr32[3] = htonl(count);

   int inner_counter = ZERO;
   for(inner_counter=ZERO; inner_counter<old_number_of_edges;
       inner_counter++) {

      if(cmpaddr(&anonymous_address,
                 &(old_edge_list[inner_counter].edge_v2),
                 128) == ZERO) {

         if(cmpaddr(&(old_edge_list[inner_counter].starting_source),
                    &(old_edge_list[inner_counter].edge_v1),
                    128) == ZERO) {

            int counter = ZERO;
            for(counter=inner_counter+1;
                counter<old_number_of_edges; counter++) {

               if((cmpaddr(&(old_edge_list[counter].starting_source),
                           &(old_edge_list[counter].edge_v1),
                           128) == ZERO) &&
                  (cmpaddr(
                     &(old_edge_list[inner_counter].starting_source),
                     &(old_edge_list[counter].edge_v1),
                     128) == ZERO) &&
                  (cmpaddr(&anonymous_address,
                     &(old_edge_list[counter].edge_v2),
                     16) == ZERO)) {

                   old_edge_list[counter].edge_v2 = anonymous_address;

               }//if((cmpaddr(&(old_edge_list[counter].starting_source)

            }//for(counter=inner_counter+1;counter<old_number_of_edges;

         }//if(cmpaddr(&(edge_list[inner_counter].starting_source), &(e

      }//if((cmpaddr(&anonymous_address, &(edge_list[inner_counter].edg

   }//for(inner_counter=ZERO; inner_counter<old_number_of_edges; inner_

}//for(count=ZERO; count<old_number_of_edges;count++) {

/*********************************************************************
*
*  Eliminate anonymous without a pair
*
*********************************************************************/

if (TROUBLESHOOT==99) {

   for(count=ZERO; count<anonymous_number;count++) {

      anonymous_address.in6_u.u6_addr32[0] = htonl(0xfe800000);
      anonymous_address.in6_u.u6_addr32[1] = 0x00000000;
      anonymous_address.in6_u.u6_addr32[2] = 0x00000000;
      anonymous_address.in6_u.u6_addr32[3] = htonl(count);

      int found = FALSE;
```

```
        int inner_counter = ZERO;
        for(inner_counter=ZERO; inner_counter<old_number_of_edges;
            inner_counter++) {

           if(cmpaddr(&anonymous_address,
                      &(old_edge_list[inner_counter].edge_v2),
                      128) == ZERO) {

              int counter = ZERO;
              for(counter=inner_counter+1;
                  counter<old_number_of_edges; counter++) {

                 if(cmpaddr(&anonymous_address,
                            &(old_edge_list[counter].edge_v2),
                            128) == ZERO) {

                    found = TRUE;

                 }//if((cmpaddr(&(old_edge_list[counter].starting_source)

              }//for(counter=inner_counter+1;counter<old_number_of_edges;

              if (found == FALSE) {

                 old_edge_list[inner_counter].edge_v2 = empty;

              }//if (found == FALSE) {

           }//if((cmpaddr(&anonymous_address, &(edge_list[inner_counter].

        }//for(inner_counter=ZERO; inner_counter<old_number_of_edges; inn

     }//for(count=ZERO; count<old_number_of_edges;count++) {

  }//if (TROUBLESHOOT==99) {

  /*********************************************************************
  *
  *  Save the edges
  *
  *********************************************************************/

  for(count=ZERO; count<old_number_of_edges;count++) {

     if (cmpaddr(&old_edge_list[count].edge_v2, &empty, 128) != ZERO) {

        int temp = add_edge(old_edge_list[count].edge_v1,
                            old_edge_list[count].edge_v2,
                            old_edge_list[count].starting_source,
                            old_edge_list[count].final_dest,
                            old_edge_list[count].hop_count);

     }//if (cmaddr(&old_edge_list[count].edge_v2, &empty, 128) != ZERO) {

  }//for(count=ZERO; count<old_number_of_edges;count++) {


} else {

   fprintf(stderr,"ANON MAIN: ERROR in loading startup files.  " \
                  "Shutting down!\n");

}//if (startup == ZERO) {

/*********************************************************************
*
* Check point if we were successfull in loading our files!!!
*
*********************************************************************/
```

```
 if (startup == ZERO) {

    // Get the current time.
    curtime = time(NULL);

    //Convert it it to local time representation.
    loctime = localtime(&curtime);
    int convert = strftime(date, DATELENGTH, "%H%M%a",loctime);
    if (convert != (7)) {

       snprintf(textbuffer, MAX_MESSAGE,"ANON MAIN: PROBLEM STARTING, " \
                                        "could not get date for file name");
       call_sys_log(textbuffer);

    }//if (num != (y)) {

    int temp = ckpnt_edges();
    if (temp != ZERO) {

       snprintf(textbuffer, MAX_MESSAGE,"ANON MAIN: Check point error");
       call_sys_log(textbuffer);
       fprintf(stderr, "ANON MAIN: We had a failure check pointing, " \
                       "check the log for error\n");

    } else {

       snprintf(textbuffer, MAX_MESSAGE,"ANON MAIN: Check point " \
                                        "successful");
       call_sys_log(textbuffer);
       fprintf(stderr,"%s\n", textbuffer);
       fprintf(stdout,"Success eliminated %i edges, shutting down!\n", \
                      (old_number_of_edges - number_of_edges));

    }//if (temp != ZERO) {

 }//if (startup == ZERO) {

/*************************************************************************
*
* Count the number of anonymous nodes left
*
*************************************************************************/

int edges_with_anon = ZERO;
int num_left = ZERO;
for(count=ZERO; count<anonymous_number; count++) {

   anonymous_address.in6_u.u6_addr32[0] = htonl(0xfe800000);
   anonymous_address.in6_u.u6_addr32[1] = 0x00000000;
   anonymous_address.in6_u.u6_addr32[2] = 0x00000000;
   anonymous_address.in6_u.u6_addr32[3] = htonl(count);

   if((cmpaddr(&anonymous_address, &(edge_list[count].edge_v2),
       16) == ZERO) ||
      (cmpaddr(&anonymous_address, &(edge_list[count].edge_v1),
       16) == ZERO)) {

      edges_with_anon++;

   }//if((cmpaddr(&anonymous_address, &(edge_list[inner_counter].edge_v2),

   int inner_counter = ZERO;
   int found         = FALSE;
   while ((inner_counter < number_of_edges) && (found == FALSE)) {

      if((cmpaddr(&anonymous_address, &(edge_list[inner_counter].edge_v2),
          128) == ZERO) ||
         (cmpaddr(&anonymous_address, &(edge_list[inner_counter].edge_v1),
```

```
          128) == ZERO)) {

          num_left++;
          found = TRUE;

      }//if((cmpaddr(&anonymous_address, &(edge_list[inner_counter].edge_v
      inner_counter++;

   }//while ((inner_counter < number_of_edges) && (found == FALSE)) {


}//for(count=ZERO; count<anonymous_number; count++) {

snprintf(textbuffer, MAX_MESSAGE,"ANON MAIN: Number of original " \
         "anonymous nodes was %i", anonymous_number);
call_sys_log(textbuffer);
fprintf(stdout,"%s\n", textbuffer);
snprintf(textbuffer, MAX_MESSAGE,"ANON MAIN: Number of anonymous nodes " \
                                 "now is %i", num_left);
call_sys_log(textbuffer);
fprintf(stdout,"%s\n", textbuffer);
snprintf(textbuffer, MAX_MESSAGE,"ANON MAIN: Number of edges with an " \
         "anonymous nodes now is %i", edges_with_anon);
call_sys_log(textbuffer);
fprintf(stdout,"%s\n", textbuffer);

/*************************************************************************
*
* Kill the sys log
*
*************************************************************************/
sendbuffer.type = SYSLOG;
snprintf(sendbuffer.text, 5, "QUIT");

sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                     sizeof(sendbuffer.text), IPC_NOWAIT);
if(sendsuccess != ZERO){

   fprintf(stderr,"ANON MAIN: ERROR: Unable to tell SYS LOG to STOP\n");
   print_IPC_error("ANON MAIN");

}//if(sendsuccess != ZERO){

/*************************************************************************
*
* Wait for the sys log to stop
*
*************************************************************************/

count = 0;
while((sys_log_pid  != ZERO) && (count <= 5)){

   printf("ANON MAIN: Waiting for sys log to finish\n");
   count++;
   sleep(1);

}//while((sys_log_pid  != ZERO) && (count <= 5)){

//Destroy the queue and clean up
int destroy = msgctl(queueid, IPC_RMID, buf);
if(destroy != ZERO){

   fprintf(stderr,"ANON MAIN: ERROR: Unable to delete the queue, error " \
                  "number = %i\n", destroy);
   print_IPC_error("ANON MAIN");

} else {

   fprintf(stderr,"ANON MAIN: Message QUEUE destroyed\n");
```

```
     }//if(destroy != ZERO){
   }// if (is_main && fork_error == FALSE) {
}//int main(int argc, char *argv[])
```

ANON_MAIN.H

```
/*****************************************************************************
 * This is the header file of the main anonymous engine
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: anon_main.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_ANON_MAIN
#define INCLUSION_GUARD_PROGRAM_ANON_MAIN


#endif //INCLUSION_GUARD_PROGRAM_ANON_MAIN
```

MAKEFILE

```
COMPILER = gcc
FINALCCFLAGS = -lpcap -lnet
INTCCFLAGS = -c
SRC=./src
BUILD=./build

default:       probe_engine alias_engine probe_builder anon_engine \
       alias_offline

all:    agr_stub probe_generator_stub pgr_stub probe_receive_stub \
       probe_main_stub

agr_stub_objects = alias_main.o ./stubs/probe_generator_stub.o sys_log.o \
       probe_utils.o ./stubs/probe_receive_stub.o probe_loader.o \
       alias_main_helper.o probe_edges.o  probe_ckpnt.o

alias_engine_objects = alias_main.o probe_generator.o sys_log.o \
       probe_utils.o probe_receive.o probe_loader.o alias_main_helper.o \
       probe_edges.o probe_ckpnt.o

alias_offline_objects = alias_offline.o sys_log.o probe_utils.o \
       probe_loader.o alias_main_helper.o probe_edges.o probe_ckpnt.o

anon_engine_objects = anon_main.o sys_log.o probe_utils.o probe_loader.o \
       probe_ckpnt.o probe_edges.o

builder_objects = probe_builder.o probe_ckpnt.o

generator_stub_objects = probe_main.o ./stubs/probe_generator_stub.o \
       sys_log.o probe_utils.o probe_receive.o  probe_loader.o \
       probe_ckpnt.o probe_stop.o probe_edges.o probe_main_helper.o

main_stub_objects = ./stubs/probe_main_stub.o probe_generator.o sys_log.o \
       probe_utils.o probe_receive.o probe_loader.o probe_ckpnt.o \
       probe_stop.o probe_edges.o probe_main_helper.o

pgr_stub_objects = probe_main.o ./stubs/probe_generator_stub.o sys_log.o \
       probe_utils.o ./stubs/probe_receive_stub.o  probe_loader.o \
       probe_ckpnt.o probe_stop.o probe_edges.o probe_main_helper.o

probe_engine_objects = probe_main.o probe_generator.o sys_log.o probe_utils.o \
       probe_receive.o probe_loader.o probe_ckpnt.o probe_stop.o \
       probe_edges.o probe_main_helper.o

receive_stub_objects = probe_main.o probe_generator.o sys_log.o probe_utils.o \
       ./stubs/probe_receive_stub.o  probe_loader.o probe_ckpnt.o \
       probe_stop.o probe_edges.o probe_main_helper.o

probe_engine: $(probe_engine_objects)
       ${COMPILER} ${FINALCCFLAGS} -o probe $(probe_engine_objects)

agr_stub: $(agr_stub_objects)
       ${COMPILER} ${FINALCCFLAGS} -o agr_stub $(agr_stub_objects)

alias_engine: $(alias_engine_objects)
       $(COMPILER) ${FINALCCFLAGS} -o alias $(alias_engine_objects)

alias_offline: $(alias_offline_objects)
       $(COMPILER) ${FINALCCFLAGS} -o alias_offline $(alias_offline_objects)

anon_engine: $(anon_engine_objects)
       ${COMPILER} ${FINALCCFLAGS} -o anonymous $(anon_engine_objects)

probe_builder: $(builder_objects)
       ${COMPILER} ${FINALCCFLAGS} -o probe_builder $(builder_objects)
```

```
probe_generator_stub: $(generator_stub_objects)
	${COMPILER} ${FINALCCFLAGS} -o probe_generator_stub \
	$(generator_stub_objects)

probe_main_stub: $(main_stub_objects)
	${COMPILER} ${FINALCCFLAGS} -o probe_main_stub $(main_stub_objects)

pgr_stub: $(pgr_stub_objects)
	${COMPILER} ${FINALCCFLAGS} -o pgr_stub $(pgr_stub_objects)

probe_receive_stub: $(receive_stub_objects)
	${COMPILER} ${FINALCCFLAGS} -o probe_receive_stub \
	$(receive_stub_objects)

%.o : ${SRC}/%.c ${SRC}/probe_main.h ${SRC}/%.h
	${COMPILER} ${INTCCFLAGS} $<

clean:
	rm -r -f *.o a.out ./stubs/*.o
```

MY_PARSE.C

```
/*****************************************************************************
 * This is the module that parses the RouteView text files for the
 * target AS
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: my_parse.c
 *
 *****************************************************************************/
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>



#define TROUBLESHOOT    0
#define BASE            10     // Base used for conversion of numbers for strnl
#define MAXFILENAME     256
#define MAXASNUMBER     5
#define MAX_LINE_LENGTH 256
#define ZERO            0
#define MAX_MESSAGE     100    // The maximum size of a message to sys logger
#define FALSE           0
#define TRUE            1

/*****************************************************************************
 *
 * Define externals and globals
 *
 *****************************************************************************/

   char  inputfile[MAXFILENAME];
   int   as_number;
   char  peerfile[] = "peerfile.txt";
   char  allfile[]  = "allfile.txt";
   char  asfile[]   = "asfile.txt";
   char  textbuffer[MAX_MESSAGE];     // Buffer to hold our message for syslog

/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/

  int read_and_parse(void){

     FILE              *source_stream;
     FILE              *peer_stream;
     FILE              *all_stream;
     FILE              *as_stream;
     // Pointer to a string for reading the file
     char              *in_string              = NULL;
     // The number of characters we read in
     int               length                  = ZERO;
     // Counter used in the loop for reading
     int               count                   = ZERO;
     // Value from converting strings to binary IPv6 addr
     int               check                   = ZERO;
     struct in6_addr   address;
```

```
snprintf(textbuffer, MAX_MESSAGE,"MY PARSE: Starting load and parse of "\
                                 "file ");
fprintf(stdout,"%s\n",textbuffer);

/*************************************************************************
*
* Open the file
*
*************************************************************************/

source_stream = fopen (inputfile, "r");
peer_stream   = fopen (peerfile, "w");
all_stream    = fopen (allfile, "w");
as_stream     = fopen (asfile, "w");

if ((source_stream == NULL) || (peer_stream == NULL) ||
    (all_stream == NULL) || (as_stream == NULL)){

    if (source_stream == NULL) {

       snprintf(textbuffer, MAX_MESSAGE,"MY PARSE: Unable to open %s "\
                                        "file", inputfile);

    } else if (peer_stream == NULL) {

       snprintf(textbuffer, MAX_MESSAGE,"MY PARSE: Unable to open %s "\
                                        "file", peerfile);

    } else if (as_stream == NULL) {

       snprintf(textbuffer, MAX_MESSAGE,"MY PARSE: Unable to open %s "\
                                        "file", asfile);

    } else {

       snprintf(textbuffer, MAX_MESSAGE,"MY PARSE: Unable to open %s "\
                                        "file", allfile);

    }//if (source_stream == NULL) {
    fprintf(stdout,"%s\n", textbuffer);
    return -1;

} else {

   /*************************************************************************
   *
   * Read a line
   *
   *************************************************************************/

            in_string           = (char *) malloc (MAX_LINE_LENGTH);
   char*    hold_in_pointer     = in_string;
   char*    temp_string         = (char *) malloc (MAX_LINE_LENGTH + 1);
   char*    hold_temp_pointer   = temp_string;
   char*    address             = NULL;
   char*    left                = NULL;
   char*    middle              = NULL;
   char*    right               = NULL;

   ssize_t  temp                = (MAX_LINE_LENGTH - 1);
   // Used for string to number
   char*    endptr              = NULL;

   if ((in_string == NULL) || (temp_string == NULL)) {

      snprintf(textbuffer, MAX_MESSAGE,"MY PARSE: ERROR: unable to "\
                                       "allocate memory for reading");
      fprintf(stdout,"%s\n", textbuffer);
```

```
   fclose(source_stream);
   fclose(peer_stream);
   fclose(all_stream);
   fclose(as_stream);
   return -1;

}//if (in_string == NULL) {

length                  = getline(&in_string, &temp, source_stream);
while (length != -1) {

   //remove any \n at the end of the line
   in_string[(length - 1)] = '\0';
   strncpy(temp_string,in_string,MAX_LINE_LENGTH);

   if (TROUBLESHOOT) {

      fprintf(stderr,"The string we read in is %s\n", in_string);

   }//if (TROUBLESHOOT) {

   char* temp_string_pointer = NULL;
   temp_string_pointer = strsep(&temp_string, "/");
   temp_string = hold_temp_pointer;
   if (TROUBLESHOOT) {

     fprintf(stderr,"temp_string_pointer = %s\n", temp_string_pointer);

   }//if (TROUBLESHOOT) {

   int done = FALSE;
   check = ZERO;
   address             = NULL;
   left                = NULL;
   middle              = NULL;
   right               = NULL;

   while ((temp_string_pointer != NULL) && (done == FALSE)){

      check = inet_pton(AF_INET6, temp_string_pointer, &address);

      if (TROUBLESHOOT) {

         fprintf(stderr, "The value of check is %i\n", check);

      }//if (TROUBLESHOOT) {
      if (check > ZERO){

         address = strsep(&in_string, " ");
         if (TROUBLESHOOT) {

            fprintf(stderr,"We matched an IPv6 address, the address "\
                           "was %s\n", address);

         }//if (TROUBLESHOOT) {
         int found = FALSE;
         int end_of_string = FALSE;
         while ((found == FALSE) && (end_of_string == FALSE)){

            middle = strsep(&in_string, " ");
            if (middle != NULL) {
               int temp_as = (int)    strtoul(middle, &endptr, 10);
               if (as_number == temp_as) {

                  right = strsep(&in_string, " ");
                  found = TRUE;
                  if (TROUBLESHOOT) {

                    fprintf(stderr,"We found it and now have right\n");
```

```
            }//if (TROUBLESHOOT) {

          } else {

             left = middle;
             middle = NULL;

             if (TROUBLESHOOT) {

                fprintf(stderr,"We did not find it left = "\
                               "middle\n");

             }//if (TROUBLESHOOT) {

          }//if (as_number == temp_as) {

        } else {

           if (TROUBLESHOOT) {

              fprintf(stderr, "Did not find the AS in this "\
                              "string\n");

           }//if (TROUBLESHOOT) {
           end_of_string = TRUE;

        }//if (middle != NULL) {

     }//while (found == FALSE) {

     if (end_of_string == FALSE) {

        if(left == NULL) {

           fprintf(as_stream,"%s %s\n", address, middle);

           if (TROUBLESHOOT){

              fprintf(stderr,"We wrote it to the as stream\n");

           }//if (TROUBLESHOOT){


        } else if(right != NULL) {

           fprintf(peer_stream, "%s %s %s %s\n", address, left, \
                                middle, right);
           fprintf(all_stream, "%s\n", left);
           fprintf(all_stream, "%s\n", right);
           if (TROUBLESHOOT){

              fprintf(stderr,"We wrote it to the peer stream\n");

           }//if (TROUBLESHOOT){

        }//if( left == NULL) {

     }//if (end_of_string != TRUE) {
     done = TRUE;

  } else {

     if (TROUBLESHOOT) {

        fprintf(stderr,"We matched an IPv4 address, tossing it "\
                       "out\n");
        fprintf(stderr,"We set done = true\n");
```

```
               }//if (TROUBLESHOOT) {
               done = TRUE;

            }//if (check > ZERO){

         }//while (in_string != NULL) {

         if (TROUBLESHOOT) {

            fprintf(stderr,"Getting the next one\n");

         }//if (TROUBLESHOOT) {

         in_string = hold_in_pointer;
         length                  = getline(&in_string, &temp, source_stream);
         temp_string = hold_temp_pointer;
         strncpy(temp_string,in_string,MAX_LINE_LENGTH);
         temp_string_pointer = strsep(&temp_string, "/");
         temp_string = hold_temp_pointer;

         if (TROUBLESHOOT) {

            fprintf(stderr,"We read length = %i and string %s\n", \
                           length,in_string);

         }//if (TROUBLESHOOT) {

       }//while (length != -1) {

       fclose(source_stream);
       fclose(peer_stream);
       fclose(all_stream);
       fclose(as_stream);
       free(hold_temp_pointer);
       free(hold_in_pointer);
       return ZERO;

    }//if ((source_stream == NULL) || (peer_stream == NULL) ||

 }//int read_and_parse(void){

/*****************************************************************************
 *
 * Main Routine
 *
 *****************************************************************************/

int main(int argc, char *argv[]){


   if (argc < 3) {

      printf("Parsing program!\n");
      printf("Usage: %s [A] [B] \n",argv[0]);
      printf(" A = Input File\n");
      printf(" B = The AS you want\n");
      printf("Example: %s myfile.txt 3257\n", argv[0]);
      printf("Thanks for playing!!\n");
      return -1;

   }//if (argc < 3) {

   if (TROUBLESHOOT==1) {

      fprintf(stderr, "I am about to do the strncpy\n");
      fprintf(stderr, "The value of argv[1] = %s\n", argv[1]);
      fprintf(stderr, "The value of argv[2] = %s\n", argv[2]);

   }//if (TROUBLESHOOT==1) {
```

```
    strncpy(inputfile, argv[1], MAXFILENAME);
    char* endptr              = NULL;    // Used for number conversion
    as_number                 = (int)    strtoul(argv[2], &endptr, 10);


    if (TROUBLESHOOT==1) {

      fprintf(stderr, "I am about to do the strncpy\n");
      fprintf(stderr, "The input file name was %s\n", inputfile);
      fprintf(stderr, "The AS number was : %i\n", as_number);

    }//if (TROUBLESHOOT==1) {

    if (as_number <= ZERO) {

       printf("The AS number is invalid\n");
       return -1;

    }//if (as_number <= ZERO) {

    int local_temp = read_and_parse();
    fprintf(stdout,"MY PARSE: Finished parsing!\n");

}//int main(int argc, char *argv[]){
```

PROBE_BUILDER.C

```
/*****************************************************************************
 * This is the main program of the probing engine and controls all operation
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_main.c
 *
 *****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <netinet/in.h>
#include <libnet.h>
#include <signal.h>
#include <unistd.h>
#include <wait.h>
#include "probe_main.h"
#include "probe_ckpnt.h"
#include "probe_builder.h"
#include <sys/socket.h>
#include <arpa/inet.h>
#include <pcap.h>


#define _GNU_SOURCE
#define TROUBLESHOOT 0
#define SOURCE_MAX 50


/*****************************************************************************
 *
 * Define externals and globals
 *
 *****************************************************************************/

   // Set by calls to the library functions to define errors encountered
   extern int errno;

   //Buffer for messages to the generator function
   struct generator_buffer  *message_to_generator;
   //Buffer for messages from receive  function
   struct receive_buffer    *message_from_receiver;
   // Pointer to the list of sources
   struct sources           *source_list;
   // Pointer to the list of edges
   struct edges             *edge_list;
   // Pointer to the global stop set
   struct global_stops      *global_stop_list;
   // Pointer to the local stop set
           void*            *probe_source_stop_list;

   struct messagebuffer     sendbuffer, recvbuffer;

   // The messaging function calls -1 an error
   const  int               IPC_ERROR       = -1;
   // The messaging function calls -1 an error
   const  int               LIBNET_ERROR    = -1;
   // The id number of queue we are using, needs to be global for all to use
          int               queueid;
   // Our IPv6 address
          char              our_ipv6_addr_name[INET6_ADDRSTRLEN+1];
   // The ipv4 address of the tunnel endpoint
          char              tunnelip[INET_ADDRSTRLEN+1];
```

```
   // The total number of sources for our probes
   int  number_of_sources        = ZERO;
   // The total number of edges found
   int  number_of_edges          = ZERO;
   // The number in the global stop table
   int  number_of_global_stops   = ZERO;
   // Used to store the pid of the probe generator
   int  probe_generator_pid      = ZERO;
   // Used to store the pid of the sys log facility
   int  sys_log_pid              = ZERO;
   // Used to store the pid of the probe receiver
   int  probe_receive_pid        = ZERO;
   // The number we are using in the anonymous nodes
   unsigned int  anonymous_number  = ZERO;


/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/


void call_sys_log(char* in_string){

    fprintf(stderr,"%s\n", in_string);


}//void call_sys_log(char* in_string){


/*****************************************************************************
*
* Function to compare two in6_addr
*
*****************************************************************************/

int cmpaddr(const struct in6_addr *source, const struct in6_addr *destination,
            const int mask) {

  int   count = ZERO;
  int   type  = ZERO;
  int   max   = ZERO;

  if (mask == 128) {

     max  = 4;
     type = 32;

  } else if ((mask%32) == ZERO) {

     type = 32;
     max  = mask/type;

  } else if ((mask%16) == ZERO) {

     type = 16;
     max  = mask/type;

  } else if ((mask%8) == ZERO) {

     type = 8;
     max  = mask/type;

  } else {

     return BADMASK;
```

```
  }//if (mask == 128) {


  if (type == 32) {

     for (count=0;count<max;count++){

        if (source->in6_u.u6_addr32[count] >
            destination->in6_u.u6_addr32[count]){

           return 1;

        } else if (source->in6_u.u6_addr32[count] <
                   destination->in6_u.u6_addr32[count]){

           return -1;

        }//if (source->in6_u.u6_addr32[count] != destination->in6_u.u6_addr32[c

     }//for (count=0;count<max;count++){

   } else if (type == 16) {

     for (count=0;count<max;count++){

        if (source->in6_u.u6_addr16[count] >
            destination->in6_u.u6_addr16[count]){

           return 1;

        } else if (source->in6_u.u6_addr16[count] <
                   destination->in6_u.u6_addr16[count]){

           return -1;

        }//if (source->in6_u.u6_addr16[count] != destination->in6_u.u6_addr16[c

     }//for (count=0;count<max;count++){


   } else {

     for (count=0;count<max;count++){

        if (source->in6_u.u6_addr8[count] >
            destination->in6_u.u6_addr8[count]){

           return 1;

        } else if (source->in6_u.u6_addr8[count] <
                   destination->in6_u.u6_addr8[count]){

           return -1;

        }//if (source->in6_u.u6_addr8[count] != destination->in6_u.u6_addr8[cou

     }//for (count=0;count<max;count++){


   }//if (type == 32) {

  return 0;

}//int cmpaddr(const struct in6_addr *source, const struct in6_addr *destinatio

/*****************************************************************************
 *
 * Main function
 *
```

```
 ******************************************************************************/

int main(int argc, char *argv[]){

   /******************************************************************************
   *
   * Define variables needed by main
   *
   ******************************************************************************/


   // Used by main to store destination addresses
   struct in6_addr          destination;
   // Hop limit used by main
   int                      hop_limit     = ZERO;
   // Used by main to store source addresses
   struct in6_addr          source;
   // Used by main to store the source port number
   int                      source_port   = ZERO;
   // Buffer to hold our messages for sys-log
   char                     textbuffer[MAX_MESSAGE];
   int                      choice       = ZERO;
   int                      done         = PLZCONTINUE;
   int                      count        = ZERO;
   int                      check        = ZERO;
   char                     tempip[INET6_ADDRSTRLEN+1];
   struct in6_addr          destinations[SOURCE_MAX];
   int                      num_dest     = ZERO;

   printf("This program will set up the probe generator files so it can " \
          "start!\n");
   printf("Note: Very little error checking is done here, make sure you " \
          "have at least two sources.\n");
   printf("The maximum number of sources and destinations that are not " \
          "sources is %i\n", SOURCE_MAX);

   source_list = (struct sources*) malloc(SOURCE_MAX * sizeof(struct sources));

   if (source_list != NULL){

      count = ZERO;
      while (done == PLZCONTINUE) {

         printf("Please enter source number %i address\n", (count+1));
         check = ZERO;
         while (check <= ZERO) {
            printf("IPv6 address: ");
            scanf("%46s", tempip);
            printf("\n");
            check = inet_pton(AF_INET6, tempip, &source);
            if (check <= ZERO){

               fprintf(stderr,"Error in the address passed in, do again\n");
               fprintf(stderr,"Input string = %s\n", tempip);

            }//if (check != ZERO){
            source_list[count].address                   = source;
            source_list[count].hop_count                 = 1;
            source_list[count].stop_next                 = NULL;
            source_list[count].remaining_probe_counter   = ZERO;
            source_list[count].h_value                   = ZERO;
            source_list[count].number_of_values          = ZERO;
            int temp = ZERO;
            for(temp=ZERO;temp<MAX_HOPS;temp++) {

               source_list[count].hop_length[temp] = ZERO;

            }//for(temp=ZERO;temp<MAX_HOPS;temp++) {
```

```
   }// while (check != ZERO) {
   printf("Enter another source?\n");
   printf("Yes                  =  1\n");
   printf("No                   =  2\n");
   scanf ("%d", &choice);
   printf("\n");
   if ((choice == 2) || (count >= (SOURCE_MAX-1))){

      if (count < 1) {

         printf("You do not have enough sources, operation aborted!\n");
         return -1;

      }//if (count < 1) {

      done = STOP;

   } else {

      count++;

   }//if (choice == 2){


}//while (done == PLZCONTINUE) {

/*************************************************************************
*
* Get destinations that are not the sources
*
*************************************************************************/
done = PLZCONTINUE;
num_dest = ZERO;
printf("Do you want to enter any destinations that are not a source?\n");
printf("Yes                   =  1\n");
printf("No                    =  2\n");
scanf ("%d", &choice);
printf("\n");
if (choice == 2) {

   done = STOP;

}//if (choice == 2) {
while (done == PLZCONTINUE) {

   printf("Please enter destination number %i address\n", (num_dest+1));
   check = ZERO;
   while (check <= ZERO) {
      printf("IPv6 address: ");
      scanf("%46s", tempip);
      printf("\n");
      check = inet_pton(AF_INET6, tempip, &source);
      if (check <= ZERO){

         fprintf(stderr,"Error in the address passed in, do again\n");
         fprintf(stderr,"Input string = %s\n", tempip);

      }//if (check != ZERO){
      destinations[num_dest] = source;

   }// while (check != ZERO) {
   printf("Enter another destination?\n");
   printf("Yes                   =  1\n");
   printf("No                    =  2\n");
   scanf ("%d", &choice);
   printf("\n");
   if ((choice == 2) || (num_dest >= (SOURCE_MAX-1))){

      done = STOP;
```

```
   } else {

      num_dest++;

   }//if (choice == 2){

}//while (done = PLZCONTINUE) {
num_dest++;
number_of_sources = count+1;
int temp = ZERO;
for(temp=ZERO;temp<number_of_sources;temp++){

   struct remaining_probes *temp_probe_pointer;
   struct remaining_probes *previous_probe_pointer;

   for(count=ZERO;count<number_of_sources;count++) {

      if(cmpaddr(&source_list[temp].address,
                 &source_list[count].address,128) != ZERO) {

         temp_probe_pointer = (struct remaining_probes*) \
                              malloc(sizeof(struct remaining_probes));
         if (temp_probe_pointer != NULL) {

            if(source_list[temp].remaining_probe_counter == ZERO) {

               source_list[temp].probe_next = temp_probe_pointer;
               temp_probe_pointer->address = source_list[count].address;
               temp_probe_pointer->next = NULL;
               source_list[temp].remaining_probe_counter++;
               previous_probe_pointer = temp_probe_pointer;

            } else {

               previous_probe_pointer->next = temp_probe_pointer;
               temp_probe_pointer->address = source_list[count].address;
               temp_probe_pointer->next = NULL;
               source_list[temp].remaining_probe_counter++;
               previous_probe_pointer = temp_probe_pointer;

            }//if(source_list[count].number_of_remaining_probes == ZERO)

         } else {

            fprintf(stderr,"ERROR: Unable to allocate memory for probe" \
                           " list\n");
            return -1;

         }//if (temp_probe_pointer == NULL) {

      }//if(cmpaddr(&source_list[temp].address,&source_list[count],128) !

   }//for(count=ZERO;count<number_of_sources;count++) {

   for(count=ZERO;count<num_dest;count++){

      temp_probe_pointer = (struct remaining_probes*) \
                           malloc(sizeof(struct remaining_probes));
      if (temp_probe_pointer != NULL) {

         previous_probe_pointer->next = temp_probe_pointer;
         temp_probe_pointer->address = destinations[count];
         temp_probe_pointer->next = NULL;
         source_list[temp].remaining_probe_counter++;
         previous_probe_pointer = temp_probe_pointer;

      } else {
```

```
               fprintf(stderr,"ERROR: Unable to allocate memory for probe " \
                              "list\n");
               return -1;

            }//if (temp_probe_pointer == NULL) {

         }//for(count=ZERO;count<num_dest;count++){


      }//for(temp=ZERO;temp<number_of_sources;temp++){

      anonymous_number       = 1;
      global_stop_list       = NULL;
      edge_list              = NULL;
      number_of_edges        = ZERO;
      number_of_global_stops = ZERO;

      if (count >= 1) {

         int temp = ckpt();
         if (temp != ZERO) {

            printf("We had a failure creating the files\n");

         } else {

            printf("Success, all files created, old files have extension " \
                   ".bak\n");

         }//if (temp != ZERO) {

      }//if (count >= 1) {


   } else {

      fprintf(stderr,"ERROR: We could not allocate memory for all the " \
                     "sources\n");
      return -1;

   }//if (source_list != NULL){
}//int main(int argc, char *argv[])
```

PROBE_BUILDER.H

```
/*****************************************************************************
 * This is the header file of the probe builder program
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_builder.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_PROBE_BUILDER
#define INCLUSION_GUARD_PROGRAM_PROBE_BUILDER


#endif //INCLUSION_GUARD_PROGRAM_PROBE_BUILDER
```

PROBE_CKPNT.C

```
/*****************************************************************************
 * This is the module that check points all the data for the system
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_ckpnt.c
 *
 *****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <netinet/in.h>
#include <libnet.h>
#include <time.h>
#include "probe_main.h"
#include "probe_ckpnt.h"


#define TROUBLESHOOT 0               // 1 = sources 2 = edges
                                     // 3 = stop 4 = probe list
#define CKPNTLENGTH  40

/*****************************************************************************
 *
 * Define externals and globals
 *
 *****************************************************************************/

   // Set by calls to the library functions to define errors encountered
   extern int errno;


          // Buffer to hold our message for syslog
          char              textbuffer[MAX_MESSAGE];

      // The current time in system format
      time_t                  curtime;
      struct tm               *loctime;
      // Used to hold the current date
      char                    date[DATELENGTH];

/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/

  int ckpnt_sources(void) {

     FILE *probe_source_stream;
     // name of the probe sources file
     char probe_source_filename[]  = "probe_source.txt";
     // name of the probe sources backup file
     char probe_source_filebak[CKPNTLENGTH];
     snprintf(probe_source_filebak, CKPNTLENGTH,"probe_source%s.bak", date);

     // Pointer to a string for writing to the file
     char *out_string             = NULL;
     // Counter used in the loop for reading
     int  count                   = ZERO;

     snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Starting check pointing" \
                                    " of sources");
```

```
call_sys_log(textbuffer);

/*************************************************************************
*
* Back up old check point file
*
*************************************************************************/

int ren = rename(probe_source_filename, probe_source_filebak);

if ((ren < ZERO) && (errno != NO_ORIGINAL_FILE)) {

    snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Unable to make " \
             "probe_source backup file! errno = %i", errno);
    call_sys_log(textbuffer);
    return -1;

} else if (errno == NO_ORIGINAL_FILE) {

    snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: No original file to " \
                                     "backup for probe_edge.txt");
    call_sys_log(textbuffer);
    if (TROUBLESHOOT==1) {

       fprintf(stderr, "%s\n", textbuffer);

    }//if (TROUBLESHOOT==1) {

} else {

   if (TROUBLESHOOT==1) {

      fprintf(stderr, "We backed up %s to %s \n", probe_source_filename,\
                       probe_source_filebak);

   }//if (TROUBLESHOOT==1) {

}//if (ren < ZERO) {

/*************************************************************************
*
* Create the space for the ourstring
*
*************************************************************************/

out_string          = (char *) malloc (MAX_LINE_LENGTH);

if (out_string == NULL){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: unable to " \
                                    "allocate memory for strings");
   call_sys_log(textbuffer);
   return -1;

}//if (in_string == NULL){

/*************************************************************************
*
* Open the sources file
*
*************************************************************************/
probe_source_stream = fopen (probe_source_filename, "w");

if (probe_source_stream == NULL) {

    snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Unable to open " \
                                     "probe_source.txt file");
    call_sys_log(textbuffer);
    return -1;
```

```
} else {

   /*********************************************************************
   *
   * Write the number of sources in the file
   *
   *********************************************************************/

   fprintf (probe_source_stream, "%i\n", number_of_sources);

   if (TROUBLESHOOT==1) {

      fprintf(stderr, "We wrote number_of_sources %i to the sources " \
                      "file: %s\n", number_of_sources, \
                      probe_source_filename);

   }//if (TROUBLESHOOT==1) {

   /*********************************************************************
   *
   * Write the sources
   *
   *********************************************************************/

   for (count=ZERO; count < number_of_sources; count++){

      /*********************************************************************
      *
      * write source address
      *
      *********************************************************************/

      if (TROUBLESHOOT==1) {

         fprintf(stderr, "About to convert!\n");

      }//if (TROUBLESHOOT==1) {

      if (inet_ntop(AF_INET6, &(source_list[count].address) , out_string,
                    INET6_ADDRSTRLEN) == NULL){

         if (TROUBLESHOOT==1) {

            fprintf(stderr, "convert done!\n");

         }//if (TROUBLESHOOT==1) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: problem " \
                  "converting source %i address for " \
                  "probe_source.txt", count);
         call_sys_log(textbuffer);
         free(out_string);
         fclose(probe_source_stream);
         return -1;

      }//if (inet_ntop(AF_INET6, &(edge_list[count].edge_v1) , out_string,

      fprintf (probe_source_stream, "%s\n", out_string);

      if (TROUBLESHOOT==1) {

         fprintf(stderr, "We wrote source %s to the source file: %s\n", \
                         out_string, probe_source_filename);

      }//if (TROUBLESHOOT==1) {

      /*********************************************************************
      *
```

```
* Write hop count
*
*****************************************************************/

fprintf (probe_source_stream, "%i\n", source_list[count].hop_count);

if (TROUBLESHOOT==1) {

   fprintf(stderr, "We wrote hop_count %i to the source file:" \
                   " %s\n", source_list[count].hop_count, \
                   probe_source_filename);

}//if (TROUBLESHOOT==1) {

/*****************************************************************
*
* Write h_value
*
*****************************************************************/

fprintf (probe_source_stream, "%i\n", source_list[count].h_value);

if (TROUBLESHOOT==1) {

   fprintf(stderr, "We wrote h_value %i to the source file: %s\n", \
                 source_list[count].h_value, probe_source_filename);

}//if (TROUBLESHOOT==1) {

/*****************************************************************
*
* Write number_of_values
*
*****************************************************************/

fprintf (probe_source_stream, "%i\n", \
         source_list[count].number_of_values);

if (TROUBLESHOOT==1) {

   fprintf(stderr, "We wrote number_of_values %i to the source " \
                   "file: %s\n", \
                   source_list[count].number_of_values, \
                   probe_source_filename);

}//if (TROUBLESHOOT==1) {

/*****************************************************************
*
* Write the hop_lengths
*
*****************************************************************/

int counter = ZERO;
for (counter=ZERO; counter < MAX_HOPS; counter++) {

   /*************************************************************
    *
    * write the value of hop length
    *
    *************************************************************/
    fprintf (probe_source_stream, "%i\n", \
             source_list[count].hop_length[counter]);

    if (TROUBLESHOOT==1) {

       fprintf(stderr, "We wrote the hop length %i for %i hop " \
                       "length for count %i\n", \
                       source_list[count].hop_length[counter], \
```

```
                                counter, count);

            }//if (TROUBLESHOOT==1) {


         }// for (counter=ZERO; counter < MAX_HOPS; counter++) {


      }//for (count=ZERO; count < number_of_sources; count++){

      free(out_string);
      fclose(probe_source_stream);

      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Finished check " \
                                       "pointing sources ");
      call_sys_log(textbuffer);
      return ZERO;


   }//if (probe_source_stream == NULL) {

}//int ckpnt_sources(void) {


int ckpnt_edges(void){

   FILE *probe_edges_stream;
   // name of the probe edges file
   char probe_edges_filename[]  = "probe_edges.txt";
   // name of the probe edges backup file
   char probe_edges_filebak[CKPNTLENGTH];
   snprintf(probe_edges_filebak, CKPNTLENGTH,"probe_edges%s.bak", date);

   // Pointer to a string for writing to the file
   char *out_string             = NULL;
   // Counter used in the loop for reading
   int  count                   = ZERO;

   snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Starting check pointing " \
                                    "of edges");
   call_sys_log(textbuffer);

   /*************************************************************************
   *
   * Back up old check point file
   *
   *************************************************************************/

   int ren = rename(probe_edges_filename, probe_edges_filebak);

   if ((ren < ZERO) && (errno != NO_ORIGINAL_FILE)) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Unable to make " \
                "probe_edge backup file! errno = %i", errno);
       call_sys_log(textbuffer);
       return -1;

   } else if (errno == NO_ORIGINAL_FILE) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: No original file to " \
                                        "backup for probe_edge.txt");
       call_sys_log(textbuffer);
       if (TROUBLESHOOT==2) {

          fprintf(stderr, "%s\n", textbuffer);

       }//if (TROUBLESHOOT==2) {

   } else {
```

```
   if (TROUBLESHOOT==2) {

      fprintf(stderr, "We backed up %s to %s \n", probe_edges_filename, \
                       probe_edges_filebak);

   }//if (TROUBLESHOOT==2) {

}//if (ren < ZERO) {

/*************************************************************************
*
* Create the space for the ourstring
*
*************************************************************************/

out_string          = (char *) malloc (MAX_LINE_LENGTH);

if (out_string == NULL){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: unable to " \
                                    "allocate memory for strings");
   call_sys_log(textbuffer);
   return -1;

}//if (in_string == NULL){

/*************************************************************************
*
* Open the edge file
*
*************************************************************************/
probe_edges_stream = fopen (probe_edges_filename, "w");

if (probe_edges_stream == NULL) {

    snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Unable to open " \
                                     "probe_edges.txt file");
    call_sys_log(textbuffer);
    return -1;

} else {


   /*********************************************************************
   *
   * This file contains the anonymous node number were working with for
   * use in marking anonymous nodes.  The first number in this file is
   * that number.
   *
   *********************************************************************/

   fprintf (probe_edges_stream, "%x\n", anonymous_number);

   if (TROUBLESHOOT==2) {

      fprintf(stderr, "We wrote anonymous number %x to the edges file:" \
                      " %s\n", anonymous_address.in6_u.u6_addr32[3], \
                      probe_edges_filename);

   }//if (TROUBLESHOOT==2) {


   /*********************************************************************
   *
   * Write the number of edge structs in the file
   *
   *********************************************************************/
```

```
fprintf (probe_edges_stream, "%i\n", number_of_edges);

if (TROUBLESHOOT==2) {

   fprintf(stderr, "We wrote number_of_edges %i to the edges file: " \
                   "%i\n", number_of_edges, probe_edges_filename);

}//if (TROUBLESHOOT==2) {

/*********************************************************************
*
* Write the edges
*
*********************************************************************/

for (count=ZERO; count < number_of_edges; count++){

   /*********************************************************************
   *
   * write edge v1
   *
   *********************************************************************/

   if (TROUBLESHOOT==2) {

      fprintf(stderr, "About to convert!\n");

   }//if (TROUBLESHOOT==2) {

   if (inet_ntop(AF_INET6, &(edge_list[count].edge_v1) , out_string,
                 INET6_ADDRSTRLEN) == NULL){

      if (TROUBLESHOOT==2) {

         fprintf(stderr, "convert done!\n");

      }//if (TROUBLESHOOT==2) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: problem " \
               "converting edge_v1 %i address for probe_edges.txt", \
               count);
      call_sys_log(textbuffer);
      free(out_string);
      fclose(probe_edges_stream);
      return -1;

   }//if (inet_ntop(AF_INET6, &(edge_list[count].edge_v1) , out_string,

   fprintf (probe_edges_stream, "%s\n", out_string);

   if (TROUBLESHOOT==2) {

      fprintf(stderr, "We wrote edge %s to the edges file: %s\n", \
                      out_string, probe_edges_filename);

   }//if (TROUBLESHOOT==2) {

   /*********************************************************************
   *
   * Write edge v2
   *
   *********************************************************************/

   if (inet_ntop(AF_INET6, &(edge_list[count].edge_v2) , out_string,
                 INET6_ADDRSTRLEN) == NULL){

      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: problem " \
               "converting edge_v2 %i address for probe_edges.txt", \
               count);
```

```
   call_sys_log(textbuffer);
   free(out_string);
   fclose(probe_edges_stream);
   return -1;

}//if (inet_ntop(AF_INET6, &(edge_list[count].edge_v2) , out_string,

fprintf (probe_edges_stream, "%s\n", out_string);

if (TROUBLESHOOT==2) {

   fprintf(stderr, "We wrote edge %s to the edges file: %s\n", \
                   out_string, probe_edges_filename);

}//if (TROUBLESHOOT==2) {

/*****************************************************************
*
* Write edge starting source
*
*****************************************************************/

if (inet_ntop(AF_INET6, &(edge_list[count].starting_source),
              out_string, INET6_ADDRSTRLEN) == NULL){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: problem " \
            "converting edge starting %i address for " \
            "probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(out_string);
   fclose(probe_edges_stream);
   return -1;

}//if (inet_ntop(AF_INET6, &(edge_list[count].starting_source) , out

fprintf (probe_edges_stream, "%s\n", out_string);

if (TROUBLESHOOT==2) {

   fprintf(stderr, "We wrote edge %s to the edges file: %s\n", \
                   out_string, probe_edges_filename);

}//if (TROUBLESHOOT==2) {

/*****************************************************************
*
* Write final dest
*
*****************************************************************/

if (inet_ntop(AF_INET6, &(edge_list[count].final_dest) , out_string,
              INET6_ADDRSTRLEN) == NULL){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: problem " \
            "converting edge starting %i address for " \
            "probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(out_string);
   fclose(probe_edges_stream);
   return -1;

}//if (check <= ZERO){

fprintf (probe_edges_stream, "%s\n", out_string);

if (TROUBLESHOOT==2) {

   fprintf(stderr, "We wrote edge %s to the edges file: %s\n", \
                   out_string, probe_edges_filename);
```

```
         }//if (TROUBLESHOOT==2) {

         /*****************************************************************
         *
         * Write hop count
         *
         *****************************************************************/

         fprintf (probe_edges_stream, "%i\n", edge_list[count].hop_count);

         if (TROUBLESHOOT==2) {

            fprintf(stderr, "We wrote hop_count %i to the edges file: %s\n",\
                            edge_list[count].hop_count, probe_edges_filename);

         }//if (TROUBLESHOOT==2) {


      }//for (count=ZERO; count < number_of_edges; count++){

      free(out_string);
      fclose(probe_edges_stream);

      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Finished check " \
                                       "pointing edges ");
      call_sys_log(textbuffer);
      return ZERO;

   }//if (probe_edges_stream == NULL) {

}//int ckpnt_edges(void){

int ckpnt_stop_list(void){

   FILE *probe_stop_stream;
   // name of the probe stop file
   char probe_stop_filename[]   = "probe_stop.txt";
   // name of the probe stop backup file
   char probe_stop_filebak[CKPNTLENGTH];
   snprintf(probe_stop_filebak, CKPNTLENGTH,"probe_stop%s.bak", date);

   // Pointer to a string for writing to the file
   char *out_string             = NULL;
   // Counter used in the loop for reading
   int  count                   = ZERO;

   snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Starting check point of" \
                                    " stop list ");
   call_sys_log(textbuffer);

   /*****************************************************************
   *
   * Back up old check point file
   *
   *****************************************************************/

   int ren = rename(probe_stop_filename, probe_stop_filebak);

   if ((ren < ZERO) && (errno != NO_ORIGINAL_FILE)) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Unable to make " \
                "probe_stop backup file! errno = %i", errno);
       call_sys_log(textbuffer);
       return -1;

   } else if (errno == NO_ORIGINAL_FILE) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: No original file to " \
```

```
                                    "backup for probe_stop.txt");
    call_sys_log(textbuffer);
    if (TROUBLESHOOT==3) {

       fprintf(stderr, "%s\n", textbuffer);

    }//if (TROUBLESHOOT==3) {

} else {

   if (TROUBLESHOOT==3) {

      fprintf(stderr, "We backed up %s to %s \n", probe_stop_filename, \
                       probe_stop_filebak);

   }//if (TROUBLESHOOT==3) {

}//if (ren < ZERO) {

/*****************************************************************************
*
* Open the stop file
*
*****************************************************************************/
probe_stop_stream = fopen (probe_stop_filename, "w");

if (probe_stop_stream == NULL) {

    snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Unable to create " \
                                     "probe_stop.txt load file");
    call_sys_log(textbuffer);
    return -1;

} else {

   /**************************************************************************
   *
   * Create the space for the ourstring
   *
   **************************************************************************/
   out_string           = (char *) malloc (MAX_LINE_LENGTH);
   if (out_string == NULL){

      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: unable to " \
                                       "allocate memory for strings");
      call_sys_log(textbuffer);
      fclose(probe_stop_stream);
      return -1;

   }//if (in_string == NULL){

   /**************************************************************************
   *
   * Write the number of global stops in the file
   *
   **************************************************************************/

   fprintf (probe_stop_stream, "%i\n", number_of_global_stops);

   if (TROUBLESHOOT==3) {

      fprintf(stderr, "We wrote number_of_global_stops %i to the stop " \
                      "file: %s\n", number_of_global_stops, \
                      probe_stop_filename);

   }//if (TROUBLESHOOT==3) {

   /**************************************************************************
   *
```

```
* Write the stops
*
*********************************************************************/

for (count=ZERO; count < number_of_global_stops; count++){

   /*******************************************************************
   *
   * Write v1
   *
   *******************************************************************/

   if (inet_ntop(AF_INET6, &(global_stop_list[count].address),
                out_string, INET6_ADDRSTRLEN) == NULL){

      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: problem " \
               "converting v1 stop %i address for probe_stop.txt", \
               count);
      call_sys_log(textbuffer);
      free(out_string);
      fclose(probe_stop_stream);
      return -1;

   }//if (inet_ntop(AF_INET6, &(global_stop_list[count].address) , out_

   fprintf (probe_stop_stream, "%s\n", out_string);

   if (TROUBLESHOOT==3) {

      fprintf(stderr, "We wrote stop %s to the stop file: %s\n", \
                      out_string, probe_stop_filename);

   }//if (TROUBLESHOOT==3) {

   /*******************************************************************
   *
   * Write destination
   *
   *******************************************************************/

   if (inet_ntop(AF_INET6, &(global_stop_list[count].dest_address),
                out_string, INET6_ADDRSTRLEN) == NULL){

      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: problem " \
               "converting v2 stop starting %i address for " \
               "probe_stop.txt", count);
      call_sys_log(textbuffer);
      free(out_string);
      fclose(probe_stop_stream);
      return -1;

   }//if (inet_ntop(AF_INET6, &(global_stop_list[count].dest_address) ,

   fprintf (probe_stop_stream, "%s\n", out_string);

   if (TROUBLESHOOT==3) {

      fprintf(stderr, "We wrote stop %s to the stop file: %s\n", \
                      out_string, probe_stop_filename);

   }//if (TROUBLESHOOT==3) {

}//for (count=ZERO; count < number_of_global_stops; count++){

/*********************************************************************
*
* Calculate the number of stops for each source
*
*********************************************************************/
```

```
int          stops[number_of_sources];
int          probe_source = ZERO;
struct stops *temp_stop_pointer;
for (probe_source=ZERO; probe_source < number_of_sources;
     probe_source++){

   stops[probe_source] = ZERO;
   temp_stop_pointer = source_list[probe_source].stop_next;
   while (temp_stop_pointer != NULL) {

      stops[probe_source]++;
      temp_stop_pointer = temp_stop_pointer->next;

   }//while (temp_stop_pointer != NULL) {

}//for (probe_source=ZERO; probe_source < number_of_sources; probe_sour

for(probe_source=ZERO;probe_source < number_of_sources;
    probe_source++){

   /*****************************************************************
   *
   * Write the number of stop structs for this source
   *
   *****************************************************************/

   fprintf (probe_stop_stream, "%i\n", stops[probe_source]);

   if (TROUBLESHOOT==3) {

      fprintf(stderr, "We wrote number of stops %i for source %i to " \
              "the stop file: %s\n", stops[probe_source], \
               probe_source, probe_stop_filename);

   }//if (TROUBLESHOOT==3) {


   if (stops[probe_source] > ZERO){

      temp_stop_pointer = source_list[probe_source].stop_next;

      for(count=ZERO;count<stops[probe_source];count++){

         /************************************************************
         *
         * Write stop
         *
         ************************************************************/

         if (inet_ntop(AF_INET6, &(temp_stop_pointer->address),
             out_string, INET6_ADDRSTRLEN) == NULL){

            snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: " \
                     "problem converting local stop %i address for" \
                     " source %i for probe_stop.txt", count, \
                     probe_source);
            call_sys_log(textbuffer);
            free(out_string);
            fclose(probe_stop_stream);
            return -1;

         }//if (inet_ntop(AF_INET6, &(temp_stop_pointer->address) , out

         fprintf (probe_stop_stream, "%s\n", out_string);

         if (TROUBLESHOOT==3) {

            fprintf(stderr, "We wrote stop %s to the stop file: %s\n",\
                            out_string, probe_stop_filename);
```

```
            }//if (TROUBLESHOOT==3) {

            temp_stop_pointer=temp_stop_pointer->next;

          }//for(count=ZERO;count<number_of_stops;count++){

        }//if (number_of_stops > ZERO){

      }//for(probe_source=ZERO;probe_source < number_of_sources; probe_source


      free(out_string);
      fclose(probe_stop_stream);

      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Finished check point " \
                                       "of stops ");
      call_sys_log(textbuffer);
      return ZERO;

   }//if (probe_stop_stream == NULL) {

}//int ckpnt_stop_list(void){

int ckpnt_probe_list(void){

   FILE *probe_list_stream;
   // name of the probe list file
   char probe_list_filename[]   = "probe_list.txt";
   // name of the probe list backup file
   char probe_list_filebak[CKPNTLENGTH];
   snprintf(probe_list_filebak, CKPNTLENGTH,"probe_list%s.bak", date);
   // Pointer to a string for writing to the file
   char *out_string             = NULL;
   // Counter used in the loop for reading
   int  count                   = ZERO;

   snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Starting check point of" \
                                    " probe list ");
   call_sys_log(textbuffer);

   /*************************************************************************
   *
   * Back up old check point file
   *
   *************************************************************************/

   int ren = rename(probe_list_filename, probe_list_filebak);

   if ((ren < ZERO) && (errno != NO_ORIGINAL_FILE)) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Unable to make " \
                "probe_list backup file! errno = %i", errno);
       call_sys_log(textbuffer);
       return -1;

   } else if (errno == NO_ORIGINAL_FILE) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: No original file to " \
                                        "backup for probe_list.txt");
       call_sys_log(textbuffer);
       if (TROUBLESHOOT==4) {

          fprintf(stderr, "%s\n", textbuffer);

       }//if (TROUBLESHOOT==4) {

   } else {
```

```
   if (TROUBLESHOOT==4) {

      fprintf(stderr, "We backed up %s to %s \n", probe_list_filename, \
                       probe_list_filebak);

   }//if (TROUBLESHOOT==4) {

}//if (ren < ZERO) {

/*************************************************************************
*
* Open the probe list file
*
*************************************************************************/
probe_list_stream = fopen (probe_list_filename, "w");

if (probe_list_stream == NULL) {

    snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Unable to open " \
                                     "probe_list.txt file for writing");
    call_sys_log(textbuffer);
    return -1;

} else {

   if (TROUBLESHOOT==4) {

      fprintf(stderr, "We opened the file %s\n", probe_list_filename);

   }//if (TROUBLESHOOT==4) {

   /*************************************************************************
   *
   * Create the space for the ourstring
   *
   *************************************************************************/
   out_string           = (char *) malloc (MAX_LINE_LENGTH);
   if (out_string == NULL){

      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: unable to " \
                                       "allocate memory for strings");
      call_sys_log(textbuffer);
      fclose(probe_list_stream);
      return -1;

   }//if (in_string == NULL){

   if (TROUBLESHOOT==4) {

      fprintf(stderr, "We created space for the out string\n");

   }//if (TROUBLESHOOT==4) {

   /*************************************************************************
   *
   * Calculate the number of probes for each source
   *
   *************************************************************************/
   int                     probe[number_of_sources];
   int                     probe_source = ZERO;
   struct remaining_probes *temp_probe_pointer;
   if (TROUBLESHOOT==4) {

      fprintf(stderr, "About to calculate the number of probes for the " \
                      "sources\n");
      fprintf(stderr, "The number of sources = %i\n", number_of_sources);
      int temp = ZERO;
      for(temp=ZERO;temp<number_of_sources;temp++) {
```

```
      fprintf(stderr,"The value of the pointer for source %i is " \
                     "%u\n",temp, source_list[temp].probe_next);
      fprintf(stderr, "The value of number_of_remaining_probes = " \
                    "%i\n", source_list[temp].remaining_probe_counter);

   }//for(temp=ZERO;temp<number_of_sources;temp++) {

}//if (TROUBLESHOOT==4) {

for (probe_source=ZERO; probe_source < number_of_sources;
     probe_source++){

   probe[probe_source] = ZERO;
   temp_probe_pointer = source_list[probe_source].probe_next;
   while (temp_probe_pointer != NULL) {

      probe[probe_source]++;
      temp_probe_pointer = temp_probe_pointer->next;

   }//while (temp_probe_pointer != NULL) {

   if (probe[probe_source] !=
       source_list[probe_source].remaining_probe_counter) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: Number " \
               "of remaining probes and number stated did not match!");
      call_sys_log(textbuffer);
      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: Number " \
               "of remaining probes = %i for source %i", \
               probe[probe_source], probe_source);
      call_sys_log(textbuffer);
      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: " \
               "source_list[probe_source].number_of_remaining_probes" \
               " = %i for source %i", \
               source_list[probe_source].remaining_probe_counter,\
               probe_source);
      call_sys_log(textbuffer);

   }//if (probe[probe_source] != source_list[probe_source].number_of_re

}//for (probe_source=ZERO; probe_source < number_of_sources; probe_sour

if (TROUBLESHOOT==4) {

   int temp = ZERO;
   fprintf(stderr, "Calculated the number of probes for the " \
                   "sources\n", probe_list_filename);
   for (temp=ZERO;temp<number_of_sources; temp++) {

      fprintf(stderr,"The number of probes for %i is %i\n",temp,\
                     probe[temp]);

   }//for (temp=ZERO;temp<number_of_sources; temp++) {

}//if (TROUBLESHOOT==4) {

for(probe_source=ZERO;probe_source < number_of_sources;
    probe_source++) {

   /*****************************************************************
   *
   * Write the number of probe structs for this source
   *
   *****************************************************************/

   fprintf (probe_list_stream, "%i\n", probe[probe_source]);

   if (TROUBLESHOOT==4) {
```

```
            fprintf(stderr, "We wrote number of probes %i for source %i to" \
                            " the stop file: %s\n", probe[probe_source], \
                            probe_source, probe_list_filename);

         }//if (TROUBLESHOOT==4) {


         if (probe[probe_source] > ZERO){

            struct remaining_probes *temp_probe_pointer;
            temp_probe_pointer = source_list[probe_source].probe_next;

            for(count=ZERO;count<probe[probe_source];count++){

               /*************************************************************
               *
               * Write probe
               *
               *************************************************************/

               if (inet_ntop(AF_INET6, &(temp_probe_pointer->address),
                   out_string, INET6_ADDRSTRLEN) == NULL){

                  snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: ERROR: " \
                           "problem converting probe %i address for source" \
                           " %i for probe_stop.txt", count, probe_source);
                  call_sys_log(textbuffer);
                  free(out_string);
                  fclose(probe_list_stream);
                  return -1;

               }//if (inet_ntop(AF_INET6, &(temp_stop_pointer->address) , out

               fprintf (probe_list_stream, "%s\n", out_string);

               if (TROUBLESHOOT==4) {

                  fprintf(stderr, "We wrote probe %s to the list file: " \
                                  "%s\n", out_string, probe_list_filename);

               }//if (TROUBLESHOOT==4) {

               temp_probe_pointer=temp_probe_pointer->next;

            }//for(count=ZERO;count<number_of_stops;count++){

         }//if (probe[probe_source] > ZERO){

      }//for(probe_source=ZERO;probe_source < number_of_sources; probe_source

      free(out_string);
      fclose(probe_list_stream);

      snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: Finished check point " \
                                       "of probe list ");
      call_sys_log(textbuffer);
      return ZERO;

   }//if (probe_list_stream == NULL) {

}//int ckpnt_probe_list(void){

int alias_ckpnt(void) {

    // Get the current time.
    curtime = time(NULL);

    //Convert it it to local time representation.
    loctime = localtime(&curtime);
```

```
    int convert = strftime(date, DATELENGTH, "%H%M%a",loctime);
    if (convert != (7)) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: PROBLEM STARTING, " \
                                        "could not get date for file name");
       call_sys_log(textbuffer);

    }//if (num != (y)) {

   return ckpnt_edges();

}//int alias_ckpnt(void) {

int ckpt(void){

    // Get the current time.
    curtime = time(NULL);

    //Convert it it to local time representation.
    loctime = localtime(&curtime);
    int convert = strftime(date, DATELENGTH, "%H%M%a",loctime);
    if (convert != (7)) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE CKPNT: PROBLEM STARTING, " \
                                        "could not get date for file name");
       call_sys_log(textbuffer);

    }//if (num != (y)) {

   if (ckpnt_edges() != ZERO){

      return -1;

   } else if (ckpnt_stop_list() != ZERO) {

      return -1;

   } else if (ckpnt_probe_list() != ZERO) {

      return -1;

   } else if (ckpnt_sources() != ZERO) {

      return -1;

   } else {

      return ZERO;

   }//if (ckpnt_edges() != ZERO){

}//int ckpt(void){
```

PROBE_CKPNT.H

```
/*****************************************************************************
 * This is the header file for the check point program
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_ckpnt.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_PROBE_CKPNT
#define INCLUSION_GUARD_PROGRAM_PROBE_CKPNT

  int ckpt(void);
  int ckpnt_edges(void);
  int ckpnt_stop_list(void);
  int ckpnt_probe_list(void);
  int ckpnt_sources(void);
  int alias_ckpnt(void);

#endif //INCLUSION_GUARD_PROGRAM_PROBE_CKPNT
```

PROBE_EDGES.C

```
/*****************************************************************************
 * This is the module decides if an edge should be saved
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_edges.c
 *
 *****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <netinet/in.h>
#include "probe_main.h"
#include "probe_utils.h"
#include "sys_log.h"


#define TROUBLESHOOT 0         // 1 = add edge
                               // 2 = add node


/*****************************************************************************
 *
 * Define externals and globals
 *
 *****************************************************************************/

   // Set by calls to the library functions to define errors encountered
   extern int errno;

   // Buffer to hold our message for syslog
          char              textbuffer[MAX_MESSAGE];

/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/

   int add_edge(const struct in6_addr edge_v1, const struct in6_addr edge_v2,
                              const struct in6_addr starting_source,
                              const struct in6_addr final_dest,
                              const        int      hop_count ){



     // Index into the global array to locate the item
     int             middle        = ZERO;
     // The result of a comparison
     int             result        = ZERO;
     // Counter for loops
     int             count         = ZERO;
     // Used as a flag to signal we have check both directions
     int             swap          = FALSE;
     // Address we will use for comparison
     struct in6_addr temp_addr;
     // Used for searching
     struct in6_addr v1_search;
     // Used for searching
     struct in6_addr v2_search;

     if (TROUBLESHOOT==1) {
```

```
   char *out_string            = NULL;
        out_string             = (char *) malloc (MAX_LINE_LENGTH);
   fprintf(stderr, "We are starting add_edge\n");
   if (inet_ntop(AF_INET6, &edge_v1 , out_string,
                 INET6_ADDRSTRLEN) == NULL){

      fprintf(stderr," The edge_v1 address passed in was not valid!\n");

   } else {

      fprintf (stderr, "The edge_v1 address passed in was : %s\n", \
                       out_string);

   }//if (inet_ntop(AF_INET6, &edgev1 , out_string, INET6_ADDRSTRLEN) == N

   if (inet_ntop(AF_INET6, &edge_v2 , out_string,
                 INET6_ADDRSTRLEN) == NULL){

      fprintf(stderr," The edge_v2 address passed in was not valid!\n");

   } else {

      fprintf (stderr, "The edge_v2 address passed in was : %s\n", \
                       out_string);

   }//if (inet_ntop(AF_INET6, &edge_v2 , out_string, INET6_ADDRSTRLEN) ==
   free(out_string);

}//if (TROUBLESHOOT==1) {

if (cmpaddr(&(edge_v1), &(edge_v2), 128) < ZERO) {

   v1_search  = edge_v1;
   v2_search  = edge_v2;

} else {

   v1_search  = edge_v2;
   v2_search  = edge_v1;

}//if (cmpaddr(&edge_v1, &edge_v2), 128) < ZERO) {

if (number_of_edges == ZERO) {

   /*********************************************************************
   *
   * Deal with start up
   *
   *********************************************************************/

   edge_list[number_of_edges].edge_v1         = v1_search;
   edge_list[number_of_edges].edge_v2         = v2_search;
   edge_list[number_of_edges].starting_source = starting_source;
   edge_list[number_of_edges].final_dest      = final_dest;
   edge_list[number_of_edges].hop_count       = hop_count;
   number_of_edges++;
   return NOT_FOUND_AND_INSERTED;

} else {

   /*********************************************************************
   *
   * Locate the middle of the edges
   *
   *********************************************************************/

   int bottom = ZERO;
   int top    = (number_of_edges - 1);
   while (bottom <= top) {
```

```
middle = (bottom + top) / 2;
result = cmpaddr(&v1_search, &(edge_list[middle].edge_v1), 128);
if (result == ZERO){

   /************************************************************
   *
   * If we are here we found the node but need to see if the
   * destination is in the list.  First locate the limits of
   * all the nodes with this address on the list
   *
   ************************************************************/
   bottom = middle;
   if (bottom > ZERO) {

      while ((result == ZERO) && (bottom > ZERO)) {

         bottom--;
         result = cmpaddr(&v1_search, &(edge_list[bottom].edge_v1),
                          128);

      }//while (result == ZERO) {
      if (result != ZERO) {

         bottom++; // The bottom of nodes with this address is here

      }//if (result != ZERO) {

   }//if (bottom > ZERO) {

   top     = middle;
   result  = ZERO;
   if (top < (number_of_edges - 1)) {

      while ((result == ZERO) && (top <(number_of_edges -1 ))) {

         top++;
         result = cmpaddr(&v1_search, &(edge_list[top].edge_v1),
                          128);

      }//while (result == ZERO) {
      if (result != ZERO) {

         top--; // The top of nodes with this address is here

      }//if (result != ZERO) {

   }//if (top < (number_of_edges - 1)) {

   int found_it = FALSE;
   if (bottom != top) {

      for(count=bottom;count<=top;count++) {

         result = cmpaddr(&(edge_list[count].edge_v2), &v2_search,
                          128);
         if (result == ZERO) {

            return FOUND;

         } else if ((result > ZERO) && (found_it == FALSE)) {

            middle = count;
            found_it = TRUE;

         }//if (result == ZERO) {

      }//for(count=bottom;count<=top;count++) {
```

```
      if ((found_it == FALSE) && (bottom != top)) {

         middle = top + 1;

      }//if (found_it == FALSE) {

   } else {

      result = cmpaddr(&v2_search, &(edge_list[bottom].edge_v2),
                       128);
      if (result == ZERO) {

         return FOUND;

      } else if (result < ZERO) {

         middle = bottom;

      } else {

         middle = bottom + 1;

      }// if (result == ZERO) {

   }//if (bottom != top) {

   if (TROUBLESHOOT==1) {

      if (found_it == TRUE) {

         fprintf(stderr,"The value of found_it is TRUE\n");

      } else {

         fprintf(stderr,"The value of found_it is FALSE\n");

      }//if (found_it == TRUE) {
      fprintf(stderr,"The value of middle is %i\n", middle);
      fprintf(stderr,"The value of top is = %i\n", top);
      fprintf(stderr,"The value of bottom is = %i\n", bottom);
      fprintf(stderr,"The value of number_of_edges = %i\n", \
                     number_of_edges);

   }//if (TROUBLESHOOT==1) {

   for(count=number_of_edges;count>middle;count--) {

      edge_list[count].edge_v1 = edge_list[(count-1)].edge_v1;
      edge_list[count].edge_v2 = edge_list[(count-1)].edge_v2;
      edge_list[count].starting_source =
                         edge_list[(count-1)].starting_source;
      edge_list[count].final_dest =
                         edge_list[(count-1)].final_dest;
      edge_list[count].hop_count  =
                         edge_list[(count-1)].hop_count;

   }//for(count=number_of_global_stops;count>bottom;count--) {

   edge_list[middle].edge_v1         = v1_search;
   edge_list[middle].edge_v2         = v2_search;
   edge_list[middle].starting_source = starting_source;
   edge_list[middle].final_dest      = final_dest;
   edge_list[middle].hop_count       = hop_count;
   number_of_edges++;
   return NOT_FOUND_AND_INSERTED;

} else if (result < ZERO) {

   top = middle - 1;
```

```
         } else {

            bottom = middle + 1;

         }//if (result == ZERO){

      }//while (bottom <= top) {

      /************************************************************************
      *
      * If we are here we did not find need to insert it
      *
      ************************************************************************/

      if (bottom > number_of_edges) {

         edge_list[number_of_edges].edge_v1         = v1_search;
         edge_list[number_of_edges].edge_v2         = v2_search;
         edge_list[number_of_edges].starting_source = starting_source;
         edge_list[number_of_edges].final_dest      = final_dest;
         edge_list[number_of_edges].hop_count       = hop_count;
         number_of_edges++;
         return NOT_FOUND_AND_INSERTED;

      } else {

         for(count=number_of_edges;count>bottom;count--) {

               edge_list[count].edge_v1 = edge_list[(count-1)].edge_v1;
               edge_list[count].edge_v2 = edge_list[(count-1)].edge_v2;
               edge_list[count].starting_source =
                                 edge_list[(count-1)].starting_source;
               edge_list[count].final_dest = edge_list[(count-1)].final_dest;
               edge_list[count].hop_count  = edge_list[(count-1)].hop_count;

         }//for(count=number_of_edges;count>bottom;count--) {
         edge_list[bottom].edge_v1         = v1_search;
         edge_list[bottom].edge_v2         = v2_search;
         edge_list[bottom].starting_source = starting_source;
         edge_list[bottom].final_dest      = final_dest;
         edge_list[bottom].hop_count       = hop_count;
         number_of_edges++;
         return NOT_FOUND_AND_INSERTED;

      }//if (bottom > number_of_edges) {*/

   }//if (number_of_edges == ZERO) {


   if (TROUBLESHOOT==1) {

      fprintf(stderr, "We are ending add_edge\n");

   }//if (TROUBLESHOOT==1) {


}//int add_edge(const struct in6_addr v1, const struct in6_addr v2 ){

int add_node(const struct in6_addr v1) {

   /************************************************************************
   *
   * Note:  Since this routine is called after locate duplicate edge we will
   *        have the node in the list at least once and have to account for
   *        it.
   *
   ************************************************************************/
```

```
// Index into the global array to locate the item
int             middle        = ZERO;
// The result of a comparison
int             result        = ZERO;
// Counter for loops
int             count         = ZERO;
// Used as a flag to signal we have check both directions
int             swap          = FALSE;
// The number we have found
int             total         = ZERO;
// Address we will use for comparison
struct in6_addr temp_addr;


if (TROUBLESHOOT==2) {

   char *out_string             = NULL;
         out_string             = (char *) malloc (MAX_LINE_LENGTH);
   fprintf(stderr, "We are starting add_node\n");
   if (inet_ntop(AF_INET6, &v1 , out_string, INET6_ADDRSTRLEN) == NULL){

      fprintf(stderr," The address passed in was not valid!\n");

   } else {

      fprintf (stderr, "The v1 address passed in was : %s\n", out_string);

   }//if (inet_ntop(AF_INET6, &v1 , out_string, INET6_ADDRSTRLEN) == NULL)

   free(out_string);

}//if (TROUBLESHOOT==2) {

if (number_of_edges == ZERO) {

   /*********************************************************************
   *
   * If we have no edges then there is nothing to find
   *
   *********************************************************************/

   return NOT_FOUND_AND_INSERTED;

} else {

   /*********************************************************************
   *
   * Locate the middle of the edges
   *
   *********************************************************************/

   int bottom = ZERO;
   int top    = (number_of_edges - 1);
   while (bottom <= top) {

      middle = (bottom + top) / 2;
      result = cmpaddr(&v1, &(edge_list[middle].edge_v1), 128);

      if (result == ZERO){

         /*****************************************************************
         *
         * If we are here we found the node but need to see if there
         * are two since we insert first
         *
         *****************************************************************/

         total++;
         bottom = middle;
```

```
if (bottom > ZERO) {

   while (result == ZERO) {

      bottom--;
      result = cmpaddr(&v1, &(edge_list[bottom].edge_v1), 128);
      if (TROUBLESHOOT==2) {

         char *out_string             = NULL;
         out_string = (char *) malloc (MAX_LINE_LENGTH);
         fprintf(stderr, "compare the this address to passed " \
                         "in: ");
         if (inet_ntop(AF_INET6, &(edge_list[bottom].edge_v1),
                       out_string, INET6_ADDRSTRLEN) == NULL){

            fprintf(stderr," The address passed in was not " \
                           "valid!\n");

         } else {

            fprintf (stderr, "%s\n", out_string);

         }//if (inet_ntop(AF_INET6, &(edge_list[bottom].edge_v1)
         fprintf(stderr,"The value of result is %i\n", result);
         free(out_string);

      }//if (TROUBLESHOOT==2) {

      if (result == ZERO) {

         total++;
         if (total >= 2) {

            return FOUND;

         }//if (total >= 2) {

      }//if (result == ZERO) {

   }//while (result == ZERO) {
   bottom++; // The bottom of nodes with this address is here

}//if (bottom > ZERO) {

top     = middle;
result  = ZERO;
if (top < (number_of_edges - 1)) {

   while (result == ZERO) {

      top++;
      result = cmpaddr(&v1, &(edge_list[top].edge_v1), 128);
      if (result == ZERO) {

         total++;
         if (total >= 2) {

            return FOUND;

         }//if (total >= 2) {

      }//if (result == ZERO) {

   }//while (result == ZERO) {
   top--; // The top of nodes with this address is here

}//if (top < (number_of_edges - 1)) {

if (TROUBLESHOOT==2) {
```

```
        printf("The value of middle is %i\n", middle);
        printf("The value of number_of_edges = %i\n", number_of_edges);

      }//if (TROUBLESHOOT==2) {

      /*************************************************************
      *
      * If we are here we did not find another and need to check other
      * side
      *
      *************************************************************/

      bottom = ZERO;  // flag for found
      count  = ZERO;
      while (count < number_of_edges) {

         result = cmpaddr(&v1, &(edge_list[count].edge_v2), 128);
         if (result == ZERO) {

            total++;
            if (total >= 2) {

               return FOUND;

            }//if (total >= 2) {

         }//if (result == ZERO) {

         count++;

      }//while ((count < number_of_edges) && (found == ZERO)) {

      return NOT_FOUND_AND_INSERTED;

   } else if (result < ZERO) {

      top = middle - 1;

   } else {

      bottom = middle + 1;

   }//if (result == ZERO){

}//while (bottom <= top) {

/*********************************************************************
*
* If we are here we did not find it and need to check the other side
*
*********************************************************************/

bottom = ZERO;  // flag for found
count  = ZERO;
while (count < number_of_edges){

   result = cmpaddr(&v1, &(edge_list[count].edge_v2), 128);
   if (result == ZERO) {

      total++;
      if (total >= 2) {

         if (TROUBLESHOOT==2) {

            fprintf(stderr," We are about to return from second we " \
                           "did find!\n");

         }//if (TROUBLESHOOT==2) {
```

```
               return FOUND;

            }//if (total >= 2) {

         }//if (result == ZERO) {

         count++;

      }//while (count < number_of_edges) {

      if (TROUBLESHOOT==2) {

         fprintf(stderr," We are about to return from last we did not " \
                        "find!\n");

      }//if (TROUBLESHOOT==2) {

      return NOT_FOUND_AND_INSERTED;

   }//if (number_of_edges == ZERO) {

   if (TROUBLESHOOT==2) {

      fprintf(stderr, "We are ending add_node\n");

   }//if (TROUBLESHOOT==2) {

}//int add_node(const struct in6_addr v1){
```

PROBE_EDGES.H

```
/*****************************************************************************
 * This is the header file for the program that decides if we should stop
 * probing
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_edges.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_PROBE_EDGES
#define INCLUSION_GUARD_PROGRAM_PROBE_EDGES

 int add_edge(const struct in6_addr edge_v1, const struct in6_addr edge_v2,
              const struct in6_addr starting_source,
              const struct in6_addr final_dest,
              const        int      hop_count );

 int add_node(const struct in6_addr v1);

#endif //INCLUSION_GUARD_PROGRAM_PROBE_EDGES
```

PROBE_GENERATOR.C

```
/*****************************************************************************
 * This is the packet generator program of the probing engine
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_generator.c
 *
 *****************************************************************************/

#include <string.h>
#include <time.h>
#include <netinet/ip6.h>
#include <netinet/udp.h>
#include <netinet/icmp6.h>
#include "probe_main.h"
#include "probe_generator.h"
#include "probe_utils.h"

#define TROUBLESHOOT 0       // 1 = startup  2 == running
#define TUNNEL_TTL   64      // The TTL of the tunnel packet
#define SAVE_POINT   1000    // The number of probes to send before we write
                             // to the log

/*****************************************************************************
 *
 * This code to calculate udp header check sum taken from tcp dump, copyright
 * below
 *
 *
 * Copyright (c) 1988, 1989, 1990, 1991, 1993, 1994, 1995, 1996
 *      The Regents of the University of California.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that: (1) source code distributions
 * retain the above copyright notice and this paragraph in its entirety, (2)
 * distributions including binary code include the above copyright notice and
 * this paragraph in its entirety in the documentation or other materials
 * provided with the distribution, and (3) all advertising materials mentioning
 * features or use of this software display the following acknowledgement:
 * ``This product includes software developed by the University of California,
 * Lawrence Berkeley Laboratory and its contributors.'' Neither the name of
 * the University nor the names of its contributors may be used to endorse
 * or promote products derived from this software without specific prior
 * written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 *****************************************************************************/


int udp6_cksum(const struct ip6_hdr *ip6, const struct udphdr *up,
        u_int len)
{
        size_t i;
        register const u_int16_t *sp;
        u_int32_t sum;
        union {
                struct {
                        struct in6_addr ph_src;
                        struct in6_addr ph_dst;
                        u_int32_t       ph_len;
                        u_int8_t        ph_zero[3];
                        u_int8_t        ph_nxt;
                } ph;
```

```
                u_int16_t pa[20];
        } phu;

        /* pseudo-header */
        memset(&phu, 0, sizeof(phu));
        phu.ph.ph_src = ip6->ip6_src;
        phu.ph.ph_dst = ip6->ip6_dst;
        phu.ph.ph_len = htonl(len);
        phu.ph.ph_nxt = IPPROTO_UDP;

        sum = 0;
        for (i = 0; i < sizeof(phu.pa) / sizeof(phu.pa[0]); i++)
                sum += phu.pa[i];

        sp = (const u_int16_t *)up;

        for (i = 0; i < (len & ~1); i += 2)
                sum += *sp++;

        if (len & 1)
                sum += htons((*(const u_int8_t *)sp) << 8);

        while (sum > 0xffff)
                sum = (sum & 0xffff) + (sum >> 16);
        sum = ~sum & 0xffff;

        return (sum);
}

/******************************************************************************
 *
 * This code to calculate icmp header check sum taken from tcp dump, copyright
 * below
 *
 *
 * Copyright (c) 1988, 1989, 1990, 1991, 1993, 1994, 1995, 1996
 *      The Regents of the University of California.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that: (1) source code distributions
 * retain the above copyright notice and this paragraph in its entirety, (2)
 * distributions including binary code include the above copyright notice and
 * this paragraph in its entirety in the documentation or other materials
 * provided with the distribution, and (3) all advertising materials mentioning
 * features or use of this software display the following acknowledgement:
 * ``This product includes software developed by the University of California,
 * Lawrence Berkeley Laboratory and its contributors.'' Neither the name of
 * the University nor the names of its contributors may be used to endorse
 * or promote products derived from this software without specific prior
 * written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 ******************************************************************************/

int icmp6_cksum(const struct ip6_hdr *ip6, const struct icmp6_hdr *icp,
        u_int len)
{
        size_t i;
        register const u_int16_t *sp;
        u_int32_t sum;
        union {
                struct {
                        struct in6_addr ph_src;
                        struct in6_addr ph_dst;
                        u_int32_t      ph_len;
                        u_int8_t       ph_zero[3];
                        u_int8_t       ph_nxt;
                } ph;
```

```
                u_int16_t pa[20];
        } phu;

        /* pseudo-header */
        memset(&phu, 0, sizeof(phu));
        phu.ph.ph_src = ip6->ip6_src;
        phu.ph.ph_dst = ip6->ip6_dst;
        phu.ph.ph_len = htonl(len);
        phu.ph.ph_nxt = IPPROTO_ICMPV6;

        sum = 0;
        for (i = 0; i < sizeof(phu.pa) / sizeof(phu.pa[0]); i++)
                sum += phu.pa[i];

        sp = (const u_int16_t *)icp;

        for (i = 0; i < (len & ~1); i += 2)
                sum += *sp++;

        if (len & 1)
                sum += htons((*(const u_int8_t *)sp) << 8);

        while (sum > 0xffff)
                sum = (sum & 0xffff) + (sum >> 16);
        sum = ~sum & 0xffff;

        return (sum);
}

/******************************************************************************
 *
 * Main part of the probe generator, no copyrighted code ends here
 *
 ******************************************************************************/
void probe_generator() {


      /*************************************************************************
      *
      * Define constants needed by the probe_generator
      *
      *************************************************************************/

      const  int       LIBNET_ERROR     = -1;
      const  u_long    LIBNET_BIN_ERROR = 0xFFFF;


      /*************************************************************************
      *
      * Define variables needed by the probe_generator
      *
      *************************************************************************/
      //Buffer for messages to the generator function
      struct generator_buffer  *message_to_generator;
      struct messagebuffer     recvbuffer;

      // Our flag to exit
      int                      done          = PLZCONTINUE;
      // Used to hold the destination address
      char                     dstname[INET6_ADDRSTRLEN+1];
      // The value of our IPC receive op!!
      int                      rcvsuccess   = ZERO;
      // Used to hold the destination address
      char                     srcname[INET6_ADDRSTRLEN+1];
      // Buffer to hold our message to syslog
      char                     textbuffer[MAX_SIZE];
      // The value of our packet send
      int                      sendsuccess         = ZERO;
      // The number of probes sent so far
```

```
unsigned long long int   probes_sent         = ZERO;
// The number of probes sent so far
unsigned long long int   probes_sent_saved   = ZERO;
// The value of this run
int                      run_number          = ZERO;
//Declare variable to hold seconds on clock.
time_t                   seconds             = ZERO;


/*************************************************************************
 *
 * Variables for libnet
 *
 *************************************************************************/

 //Initial handle for our packet
 libnet_t                *packet_handle;
 //p_tag which allows us to modify
 libnet_ptag_t           ip_packet_tag;
 //Our Binary IPv4 address
 u_long                  src_ip;
 //Binary IPv4 address of the tunnel
 u_long                  tun_ip;
 // The length of our UDP IP packet
 u_int16_t               ip_length;
 // The destination port of our UDP packet
 u_short                 dst_prt;
 // The source port of our UDP Packet
 u_short                 src_prt;
 // The hop limit of the IPv6 packet
 u_int8_t                packethoplimit;

 struct ip6_hdr*         my_ipv6;
 struct ip6_rthdr0*      udp_route_hdr;
 struct ip6_rthdr0*      icmp_route_hdr;
 struct udphdr*          udp_source_hdr;
 struct udphdr*          udp_hdr;
 struct icmp6_hdr*       icmp_hdr;
 struct icmp6_hdr*       icmp_source_hdr;
 char                    *udp_source_str_hdr           = NULL;
 char                    *udp_str_hdr                  = NULL;
 char                    *icmp_source_str_hdr           = NULL;
 char                    *icmp_str_hdr                  = NULL;

 // Our Binary IPv6 address
 struct in6_addr         our_ipv6_addr;
 //Buffer to hold error messages from Libnet
 char                    neterrbuf[LIBNET_ERRBUF_SIZE];

 char * device = OUTDEVICE;      // Device we will send our data out on

 int                     sendsize                      = ZERO;


 /*************************************************************************
 *
 * Determine our run number for the log
 *
 *************************************************************************/

 // Get value from system clock place in seconds variable to use as seed.
 time(&seconds);

 //Convert seconds to a unsigned integer and seed.
 srand((unsigned int) seconds);

 run_number = rand();

 /*************************************************************************
```

```
 *
 * Build the payload, we do this before the packet because we need the
 * length of the message
 *
 ************************************************************************/

 // Tell people how to contact
 char             *payload         = "Thesis Research, to be removed " \
                                     "from scans please e-mail " \
                                     "XXXXXXXXXXX@nps.edu";
 // The length of our payload!!
 u_short          payload_s        = strlen(payload) + 1;

 /************************************************************************
 *
 * Build buffer for udp packet
 *
 ************************************************************************/

 int                     bufsize        = sizeof(struct ip6_hdr) +
                                          sizeof(struct ip6_rthdr0) +
                                          sizeof(struct udphdr) + payload_s;
 //Holds the data temporarly until we process it
 unsigned char           buf[bufsize];
 memset(buf, 0 , bufsize);


 /************************************************************************
 *
 * set up pointers for different packets
 *
 ************************************************************************/

 my_ipv6       = (struct ip6_hdr*) buf;
 udp_route_hdr = (struct ip6_rthdr0*)(buf + sizeof(struct ip6_hdr));
 udp_source_hdr = (struct udphdr*) (buf + sizeof(struct ip6_hdr) +
                  sizeof(struct ip6_rthdr0));
 udp_source_str_hdr   = (char *) (buf + sizeof(struct ip6_hdr) +
                  sizeof(struct ip6_rthdr0) +
                  sizeof(struct udphdr));


 udp_hdr     = (struct udphdr*) (buf + sizeof(struct ip6_hdr));
 udp_str_hdr = (char *) (buf + sizeof(struct ip6_hdr) +
                         sizeof(struct udphdr));
 icmp_hdr    = (struct icmp6_hdr*) (buf + sizeof(struct ip6_hdr));
 icmp_str_hdr = (char *) (buf + sizeof(struct ip6_hdr) +
                          sizeof(struct icmp6_hdr));
 icmp_route_hdr  = (struct ip6_rthdr0*)(buf + sizeof(struct ip6_hdr));
 icmp_source_hdr = (struct icmp6_hdr*) (buf + sizeof(struct ip6_hdr) +
                    sizeof(struct ip6_rthdr0));
 icmp_source_str_hdr  = (char *) (buf + sizeof(struct ip6_hdr) +
                         sizeof(struct ip6_rthdr0) +
                         sizeof(struct icmp6_hdr));


/************************************************************************
*
* Process requests until we are told to stop
*
************************************************************************/

snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: Starting the probe " \
                                 "generator run number %i", run_number);
call_sys_log(textbuffer);

/************************************************************************
*
* Set up libnet with the info needed and open up the session
```

```
* Convert the addresses to binary and find the address of our interface
* and print it
*
*********************************************************************/

//Open device for writing
packet_handle      = libnet_init(LIBNET_RAW4,device,neterrbuf);
//Get the our IPv4 address from the device
src_ip = libnet_get_ipaddr4(packet_handle);
int check = inet_pton(AF_INET6, our_ipv6_addr_name, &our_ipv6_addr);
//set the tunnel ip
tun_ip = libnet_name2addr4(packet_handle,tunnelip,LIBNET_DONT_RESOLVE);

if (TROUBLESHOOT==1) {

   //Print what address we are using!!
   fprintf(stderr, "The address for device %s that I will use as the" \
                   " source is:  %s\n", device, \
                   libnet_addr2name4(src_ip, LIBNET_DONT_RESOLVE));

}//if (TROUBLESHOOT==1) {


if (packet_handle == NULL) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: ERROR: Could not" \
            " open libnet session: %s\n", libnet_geterror(packet_handle));
   call_sys_log(textbuffer);
   done = STOP;

} else if (src_ip == LIBNET_ERROR) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: ERROR: Could not" \
            " get ip of device: %s", libnet_geterror(packet_handle));
   call_sys_log(textbuffer);
   done = STOP;

} else if (check <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: ERROR: Could not" \
                                    " convert our IPv6 addr");
   call_sys_log(textbuffer);
   done = STOP;

} else if (tun_ip == LIBNET_BIN_ERROR) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: ERROR: Could not" \
            " convert input IP addresses to binary: %s", \
            libnet_geterror(packet_handle));
   call_sys_log(textbuffer);
   done = STOP;

}//    if (packet_handle == NULL) {
/*********************************************************************
*
* Tell the system to ignore the quit signal for proper shutdown
*
*********************************************************************/

signal(SIGQUIT, SIG_IGN);

while (done == PLZCONTINUE) {

    //Get the message off the queue, block until one arrives
    rcvsuccess = msgrcv(queueid, (void *) &recvbuffer,
                        sizeof(struct generator_buffer), GENERATOR,
                        IPC_WAIT);

    if(rcvsuccess == IPC_ERROR) {
```

```
        snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: ERROR: " \
                                         "Message Receive failed.");
        call_sys_log(textbuffer);
        print_IPC_error("PROBE GENERATOR");
        done = STOP;

    } else {

     message_to_generator = (struct generator_buffer*) (recvbuffer.text);

       if (message_to_generator->generator_cont != PLZCONTINUE) {

          done = STOP;

       } else {


/*********************************************************************
*
* Get the address names for putting into the log and routing header
* NOTE: THIS NEEDS TO BE CHANGED SO WE DON'T ENCODE THE ADDRESSES BEFORE
* SENDING THEM HERE
*
*********************************************************************/

          inet_ntop(AF_INET6, &(message_to_generator->destination),
                    dstname, INET6_ADDRSTRLEN);
          inet_ntop(AF_INET6, &(message_to_generator->source), srcname,
                    INET6_ADDRSTRLEN);
          snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: Processing"\
                                           " packet to: %s", dstname);
          call_sys_log(textbuffer);

          if (TROUBLESHOOT==2) {

             fprintf(stderr, "message_to_generator->protocol =  %i\n",\
                              message_to_generator->protocol);

          }//if (TROUBLESHOOT==2) {


          if (message_to_generator->protocol == IPPROTO_ICMPV6 ||
              message_to_generator->protocol == (IPPROTO_ICMPV6 +
                                                 SOURCEROUTE)) {


             if (message_to_generator->protocol == (IPPROTO_ICMPV6 +
                                                    SOURCEROUTE)){

                if (TROUBLESHOOT==2) {

                   fprintf(stderr, "We are in the source route ICMPV6" \
                                   " section\n");

                }//if (TROUBLESHOOT==2) {

                memset(buf, 0 , bufsize);
                sendsize       = sizeof(struct ip6_hdr) +
                                 sizeof(struct ip6_rthdr0) +
                                 sizeof(struct icmp6_hdr) + payload_s;
                packethoplimit = message_to_generator->hop_limit;

                if (TROUBLESHOOT==2) {

                   fprintf(stderr, "Processing packet with hop limit " \
                                   "= %i\n", packethoplimit);

                }//if (TROUBLESHOOT==2) {
```

```
/*******************************************************
 *
 *  Build the payload
 *
 *******************************************************/

 memcpy(icmp_source_str_hdr, payload, payload_s);

if (TROUBLESHOOT==2) {

   fprintf(stderr, "Completed memcpy, str = %s\n", \
                   icmp_source_str_hdr);

}//if (TROUBLESHOOT==2) {

/*******************************************************
 *
 *  Build the ICMP Packet
 *
 *******************************************************/

 icmp_source_hdr->icmp6_type  = ICMP6_ECHO_REQUEST;
 icmp_source_hdr->icmp6_code  = ZERO;
 icmp_source_hdr->icmp6_cksum = ZERO;
 icmp_source_hdr->icmp6_id =
                htons(message_to_generator->port);
 icmp_source_hdr->icmp6_seq =
                htons(message_to_generator->sequence);

 if (TROUBLESHOOT==2) {

    fprintf(stderr, "Built ICMP\n");
    fprintf(stderr, "payload length = %i\n", payload_s);
    fprintf(stderr, "Length = %i\n", sendsize);

 }//if (TROUBLESHOOT==2) {


/*******************************************************
 *
 *  Build the Routing header
 *
 *******************************************************/

 icmp_route_hdr->ip6r0_nxt      = IPPROTO_ICMPV6;
 icmp_route_hdr->ip6r0_len      = 2;
 icmp_route_hdr->ip6r0_type     = ZERO;
 icmp_route_hdr->ip6r0_segleft  = 1;
 icmp_route_hdr->ip6r0_reserved = ZERO;
 icmp_route_hdr->ip6r0_slmap[0] = ZERO;
 icmp_route_hdr->ip6r0_slmap[1] = ZERO;
 icmp_route_hdr->ip6r0_slmap[2] = ZERO;
 icmp_route_hdr->ip6r0_addr[0]  =
                     message_to_generator->destination;

 if (TROUBLESHOOT==2) {

    fprintf(stderr, "Built Routing Header\n");

 }//if (TROUBLESHOOT==2) {

 /*******************************************************
 *
 *  Build the ipv6 header
 *
 *******************************************************/

 my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_flow = 0x00000060;
```

```
    my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_plen =
                          htons(sizeof(struct icmp6_hdr) +
                          sizeof(struct ip6_rthdr0) +
                          payload_s);
    my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_nxt =
                          LIBNET_IPV6_NH_ROUTING;
    my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_hlim =
                          packethoplimit;
    my_ipv6->ip6_src = our_ipv6_addr;
    // Need this for correct checksum
    my_ipv6->ip6_dst = message_to_generator->destination;

    unsigned int sum = icmp6_cksum(my_ipv6, icmp_source_hdr,
                       (sizeof(struct icmp6_hdr) +
                        payload_s));
    icmp_source_hdr->icmp6_cksum = sum;
    // Need this for correct checksum
    my_ipv6->ip6_dst = message_to_generator->source;

    if (TROUBLESHOOT==2) {

       fprintf(stderr,"The version is : 0x%4.4x\n", \
                      my_ipv6->ip6_ctlun.ip6_un2_vfc);
       fprintf(stderr, "checksum 1 = 0x%4.4x\n", sum);
       fprintf(stderr, "Built IPv6 Header\n");
       fprintf(stderr, "Hop limit = %i\n", packethoplimit);
       fprintf(stderr, "Next Header = %i\n", \
                       LIBNET_IPV6_NH_ROUTING);
       fprintf(stderr, "The string is : %s\n", \
                       icmp_source_str_hdr);
       fprintf(stderr, "The size of our buffer is %i\n", \
                       sendsize);

    }//if (TROUBLESHOOT==2) {

} else {

   if (TROUBLESHOOT==2) {

      fprintf(stderr, "Start the build of the ICMPV6 " \
                      "packet\n");

   }//if (TROUBLESHOOT==2) {

   memset(buf, 0 , bufsize);
   sendsize       = sizeof(struct ip6_hdr) +
                    sizeof(struct icmp6_hdr) + payload_s;
   packethoplimit = message_to_generator->hop_limit;

   if (TROUBLESHOOT==2) {

      fprintf(stderr, "Processing packet with hop limit " \
                      "= %i\n", packethoplimit);

   }//if (TROUBLESHOOT==2) {

   /*********************************************************
    *
    *  Build the payload
    *
    *********************************************************/

    memcpy(icmp_str_hdr, payload, payload_s);

   if (TROUBLESHOOT==2) {

      fprintf(stderr, "Completed memcpy, str = %s\n", \
                      icmp_str_hdr);
```

```
      }//if (TROUBLESHOOT==2) {

      /*****************************************************
       *
       *  Build the ICMP Packet
       *
       *****************************************************/

       icmp_hdr->icmp6_type  = ICMP6_ECHO_REQUEST;
       icmp_hdr->icmp6_code  = ZERO;
       icmp_hdr->icmp6_cksum = ZERO;
       icmp_hdr->icmp6_id =
                        htons(message_to_generator->protocol);
       icmp_hdr->icmp6_seq   =
                        htons(message_to_generator->sequence);

       if (TROUBLESHOOT==2) {

          fprintf(stderr, "Built ICMP\n");
          fprintf(stderr, "payload length = %i\n", payload_s);
          fprintf(stderr, "Length = %i\n", sendsize);

       }//if (TROUBLESHOOT==2) {

       /*****************************************************
       *
       *  Build the ipv6 header
       *
       *****************************************************/

       my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_flow = 0x00000060;
       my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_plen =
                           htons(sizeof(struct icmp6_hdr) +
                           payload_s);
       my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_nxt = IPPROTO_ICMPV6;
       my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_hlim =
                           packethoplimit;
       my_ipv6->ip6_src = our_ipv6_addr;
       // Need this for correct checksum
       my_ipv6->ip6_dst = message_to_generator->destination;

       unsigned int sum = icmp6_cksum(my_ipv6, icmp_hdr,
                          (sizeof(struct icmp6_hdr) +
                           payload_s));
       icmp_hdr->icmp6_cksum = sum;

       if (TROUBLESHOOT==2) {

          fprintf(stderr,"The version is : 0x%4.4x\n", \
                        my_ipv6->ip6_ctlun.ip6_un2_vfc);
          fprintf(stderr, "checksum 1 = 0x%4.4x\n", sum);
          fprintf(stderr, "Built IPv6 Header\n");
          fprintf(stderr, "Hop limit = %i\n", packethoplimit);
          fprintf(stderr, "Next Header = %i\n",IPPROTO_ICMP);
          fprintf(stderr, "The string is : %s\n", \
                          icmp_str_hdr);
          fprintf(stderr, "The size of our buffer is %i\n", \
                          sendsize);

       }//if (TROUBLESHOOT==2) {


    }// if (message_to_generator->protocol == (IPPROTO_ICMP + SO


 } else {

    if (message_to_generator->protocol == (IPPROTO_UDP +
                                           SOURCEROUTE)){
```

```
memset(buf, 0 , bufsize);
sendsize       = sizeof(struct ip6_hdr) +
                 sizeof(struct ip6_rthdr0) +
                 sizeof(struct udphdr) + payload_s;
src_prt        = message_to_generator->port;
packethoplimit = message_to_generator->hop_limit;
dst_prt        = message_to_generator->sequence;

if (TROUBLESHOOT==2) {

   fprintf(stderr, "Processing packet with src port = " \
                   "%i\n", src_prt);
   fprintf(stderr, "Processing packet with destination" \
                   " port = %i\n", dst_prt);
   fprintf(stderr, "Processing packet with hop limit =" \
                   " %i\n", packethoplimit);

}//if (TROUBLESHOOT==2) {

/*******************************************************
 *
 *  Build the payload
 *
 *******************************************************/

 memcpy(udp_source_str_hdr, payload, payload_s);

/*******************************************************
 *
 *  Build the UDP Packet
 *
 *******************************************************/

 udp_source_hdr->source = ntohs(src_prt);
 udp_source_hdr->dest   = ntohs(dst_prt);
 udp_source_hdr->len   = ntohs(LIBNET_UDP_H + payload_s);
 udp_source_hdr->check  = ZERO;

 if (TROUBLESHOOT==2) {

    fprintf(stderr, "Built UDP\n");
    fprintf(stderr, "payload length = %i\n", payload_s);
    fprintf(stderr, "Length = %i\n", (LIBNET_UDP_H +
                                      payload_s));

 }//if (TROUBLESHOOT==2) {

/*******************************************************
 *
 *  Build the Routing header
 *
 *******************************************************/

 udp_route_hdr->ip6r0_nxt = IPPROTO_UDP;
 udp_route_hdr->ip6r0_len = 2;
 udp_route_hdr->ip6r0_type = ZERO;
 udp_route_hdr->ip6r0_segleft = 1;
 udp_route_hdr->ip6r0_reserved = ZERO;
 udp_route_hdr->ip6r0_slmap[0] = ZERO;
 udp_route_hdr->ip6r0_slmap[1] = ZERO;
 udp_route_hdr->ip6r0_slmap[2] = ZERO;
 udp_route_hdr->ip6r0_addr[0] =
                    message_to_generator->destination;

 if (TROUBLESHOOT==2) {

    fprintf(stderr, "Built Routing Header\n");
```

```
    }//if (TROUBLESHOOT==2) {

    /*****************************************************
    *
    *  Build the ipv6 header
    *
    *****************************************************/

    my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_flow = 0x00000060;
    my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_plen =
                         htons(sizeof(struct ip6_rthdr0) +
                        sizeof(struct udphdr) + payload_s);
    my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_nxt =
                         LIBNET_IPV6_NH_ROUTING;
    my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_hlim =
                         packethoplimit;
    my_ipv6->ip6_src = our_ipv6_addr;
    // Need this for correct checksum
    my_ipv6->ip6_dst = message_to_generator->destination;

    unsigned int sum = udp6_cksum(my_ipv6, udp_source_hdr,
                       (sizeof(struct udphdr) + payload_s));
    udp_source_hdr->check  = sum;
    // Need this for correct checksum
    my_ipv6->ip6_dst = message_to_generator->source;

    if (TROUBLESHOOT==2) {

       fprintf(stderr,"The version is : 0x%4.4x\n", \
                     my_ipv6->ip6_ctlun.ip6_un2_vfc);
       fprintf(stderr, "checksum 1 = 0x%4.4x\n", sum);
       fprintf(stderr, "Built IPv6 Header\n");
       fprintf(stderr, "Hop limit = %i\n", packethoplimit);
       fprintf(stderr, "Next Header = %i\n", \
                        LIBNET_IPV6_NH_ROUTING);
       fprintf(stderr, "The string is : %s\n", \
                        udp_source_str_hdr);
       fprintf(stderr, "The size of our buffer is %i\n",\
                        sendsize);

    }//if (TROUBLESHOOT==2) {


} else {

   if (TROUBLESHOOT==2) {

      fprintf(stderr,"Starting the build of the UDP packet" \
                     "!\n");

   }//if (TROUBLESHOOT==2) {


   memset(buf, 0 , bufsize);
   sendsize       = sizeof(struct ip6_hdr) +
                    sizeof(struct udphdr) + payload_s;
   src_prt        = message_to_generator->port;
   packethoplimit = message_to_generator->hop_limit;
   dst_prt        = message_to_generator->sequence;

   if (TROUBLESHOOT==2) {

      fprintf(stderr, "Processing packet with src port " \
                      "= %i\n", src_prt);
      fprintf(stderr, "Processing packet with hop limit" \
                      " = %i\n", packethoplimit);

   }//if (TROUBLESHOOT==2) {
```

```
   /*****************************************************
    *
    *  Build the payload
    *
    *****************************************************/

    memcpy(udp_str_hdr, payload, payload_s);

   if (TROUBLESHOOT==2) {

      fprintf(stderr, "Completed memcpy, str = %s\n",
                     udp_str_hdr);

   }//if (TROUBLESHOOT==2) {

   /*****************************************************
    *
    *  Build the UDP Packet
    *
    *****************************************************/

    udp_hdr->source = ntohs(src_prt);
    udp_hdr->dest   = ntohs(dst_prt);
    udp_hdr->len    = ntohs(LIBNET_UDP_H + payload_s);
    udp_hdr->check  = ZERO;

    if (TROUBLESHOOT==2) {

       fprintf(stderr, "Built UDP\n");
       fprintf(stderr, "payload length = %i\n", payload_s);
       fprintf(stderr, "Length = %i\n", (LIBNET_UDP_H +
                       payload_s));

    }//if (TROUBLESHOOT==2) {

    /*****************************************************
    *
    *  Build the ipv6 header
    *
    *****************************************************/

    my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_flow = 0x00000060;
    my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_plen =
                  htons(sizeof(struct udphdr) + payload_s);
    my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_nxt = IPPROTO_UDP;
    my_ipv6->ip6_ctlun.ip6_un1.ip6_un1_hlim = packethoplimit;
    my_ipv6->ip6_src = our_ipv6_addr;
    // Need this for correct checksum
    my_ipv6->ip6_dst = message_to_generator->destination;

    unsigned int sum = udp6_cksum(my_ipv6, udp_hdr,
                       (sizeof(struct udphdr) + payload_s));
    udp_hdr->check  = sum;

    if (TROUBLESHOOT==2) {

       fprintf(stderr,"The version is : 0x%4.4x\n", \
                      my_ipv6->ip6_ctlun.ip6_un2_vfc);
       fprintf(stderr, "checksum 1 = 0x%4.4x\n", sum);
       fprintf(stderr, "Built IPv6 Header\n");
       fprintf(stderr, "Hop limit = %i\n", packethoplimit);
       fprintf(stderr, "Next Header = %i\n",IPPROTO_UDP);
       fprintf(stderr, "The string is : %s\n", udp_str_hdr);
       fprintf(stderr, "The size of our buffer is %i\n", \
                       sendsize);

    }//if (TROUBLESHOOT==2) {

}//if (message_to_generator->protocol == (IPPROTO_UDP + SOUR
```

```
} //if (message_to_generator->protocol == IPPROTO_ICMP) {

ip_length     = LIBNET_IPV4_H + sendsize;//The length of our ip
ip_packet_tag = libnet_build_ipv4(ip_length,ZERO,ZERO,IP_DF,
                                  TUNNEL_TTL,TUNNEL_PROTO,
                                  ZERO,src_ip,tun_ip,buf,
                                  sendsize,packet_handle,ZERO);

if (ip_packet_tag == LIBNET_ERROR) {

    snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: ERROR:"\
                              " building IP Packet: %s\n", \
                              libnet_geterror(packet_handle));
    call_sys_log(textbuffer);
    done = STOP;

} //    if (UDP_ip_packet_tag == LIBNET_ERROR) {

if (TROUBLESHOOT==2) {

    if (ip_packet_tag == LIBNET_ERROR) {

       fprintf(stderr,"PROBE GENERATOR: ERROR: building IP" \
                     " Packet: %s\n", \
                     libnet_geterror(packet_handle));

    } else {

      fprintf(stderr, "Built IP Header\n");

    }//if (ip_packet_tag == LIBNET_ERROR) {

}//if (TROUBLESHOOT==2) {

/*************************************************************
 *
 *  Always send three to make sure we are not hitting a fluke
 *
 *************************************************************/

int temp_count = ZERO;
for(temp_count=ZERO;temp_count<NUMBER_OF_PACKETS;temp_count++){

   if (ip_packet_tag != LIBNET_ERROR) {

      //Send the data!!!
      sendsuccess = libnet_write(packet_handle);
      if (sendsuccess < ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: " \
                  "ERROR: could not write dst %s error = : " \
                  "%s", dstname, \
                  libnet_geterror(packet_handle));
         call_sys_log(textbuffer);
         done = STOP;

      } else {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: " \
                  "Sent packet to %s with hop limit % i", \
                   dstname, message_to_generator->hop_limit);
         call_sys_log(textbuffer);
         probes_sent++;
         if (probes_sent >= (probes_sent_saved + SAVE_POINT)) {

            snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR"\
                     ": Run number %i sent %llu probes so " \
```

```
                                    "far", run_number, probes_sent);
                              call_sys_log(textbuffer);
                              probes_sent_saved = probes_sent;

                           }// if (probes_sent >= (probes_sent_saved + SAVE_POINT

                        }//    if (success < ZERO) {

                        if (TROUBLESHOOT==2) {

                           fprintf(stderr, "Completed the send, success = %i\n",\
                                           sendsuccess);

                        }//if (TROUBLESHOOT==2) {

                     }//for(temp_count=ZERO;temp_count<3;temp_count++){

                  }//if (ip_packet_tag != LIBNET_ERROR) {

                  libnet_clear_packet(packet_handle);

               }//if (message_to_generator->generator_cont != PLZCONTINUE) {

            }//if(rcvsuccess == IPC_ERROR) {

         }//while (done == PLZCONTINUE) {

         libnet_destroy(packet_handle);  // Shut down our socket
         snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: Run number %i sent" \
                                          " %llu probes", run_number, probes_sent);
         call_sys_log(textbuffer);
         snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: Stopping run number" \
                                        " %i for the probe generator", run_number);
         call_sys_log(textbuffer);

}//void probe_generator() {
```

PROBE_GENERATOR.H

```
/*****************************************************************************
 * This is the header file of the main program of the probing engine
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_generator.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_PROBE_GENERATOR
#define INCLUSION_GUARD_PROGRAM_PROBE_GENERATOR

    void probe_generator();

#endif //INCLUSION_GUARD_PROGRAM_PROBE_GENERATOR
```

PROBE_GENERATOR_STUB.C

```
/*****************************************************************************
 * This is the packet generator program of the probing engine
 * (////////////////////////////////////STUB////////////////////////////////////)
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_generator.c
 *
 *****************************************************************************/

#include "../src/probe_main.h"
#include "../src/probe_generator.h"
#include "../src/probe_utils.h"

void probe_generator() {


      /*****************************************************************************
      *
      * Define variables needed by the probe_generator
      *
      *****************************************************************************/
      //Buffer for messages to the generator function
      struct generator_buffer  *message_to_generator;
      struct messagebuffer     recvbuffer;
      // Our flag to exit
      int                      done          = PLZCONTINUE;
      // The value of our receive op!!
      int                      rcvsuccess    = ZERO;
      // Buffer to hold our message to syslog
      char                     textbuffer[MAX_MESSAGE];

      /*****************************************************************************
      *
      * Process requests until we are told to stop
      *
      *****************************************************************************/

      signal(SIGQUIT, SIG_IGN);

      snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: Starting the probe"\
                                       " generator");
      printf("%s\n",textbuffer);

      while (done == PLZCONTINUE) {

          //Get the message off the queue, block until one arrives
          rcvsuccess = msgrcv(queueid, (void *) &recvbuffer,
                              sizeof(struct generator_buffer),
                              GENERATOR, IPC_WAIT);

          if(rcvsuccess == IPC_ERROR) {

              snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: Message "\
                                               "Receive failed.");
              printf("%s\n", textbuffer);
              print_IPC_error("PROBE GENERATOR");
              done = STOP;

          } else {

             printf("////////////////////////////GENERATE NEW PACKET//////"\
                    "//////////////////\n");
             printf("Message receive was successful\n");
```

```
            printf("for queue id = %d\n", queueid);
            //Print the number of bytes received,it is equal to return value.
            printf("Bytes received = %d\n", rcvsuccess);

            message_to_generator =
                                (struct generator_buffer*) (recvbuffer.text);

            printf("Processing is to : ");
            if (message_to_generator->generator_cont == PLZCONTINUE) {

               printf("continue.\n");

            } else {

               printf("stop.\n");
               done = STOP;

            }//if (message_to_generator->generator_cont == PLZCONTINUE) {

            char hostname[INET6_ADDRSTRLEN + 1];

            inet_ntop(AF_INET6, &(message_to_generator->source) , hostname,
                      INET6_ADDRSTRLEN);
            printf("Source address received was: %s\n", hostname);
            inet_ntop(AF_INET6, &(message_to_generator->destination) ,
                      hostname, INET6_ADDRSTRLEN);
            printf("Destination address received was: %s\n", hostname);
            printf("The hop limit received was: %i\n", \
                    message_to_generator->hop_limit);
            if (message_to_generator->protocol == IPPROTO_ICMPV6) {

               printf("The protocol received was: ICMPV6\n");

            } else if (message_to_generator->protocol ==
                       (IPPROTO_ICMPV6 + SOURCEROUTE)) {

              printf("The protocol received was: ICMPV6 plus Source Route\n");

            } else if (message_to_generator->protocol ==
                       (IPPROTO_UDP + SOURCEROUTE)) {

               printf("The protocol received was: UDP plus Source Route\n");

            } else {

               printf("The protocol received was: UDP\n");

            }
            printf("The protocol number received was: %i\n",\
                    message_to_generator->protocol);
            printf("The port/id received was: %i\n",\
                    message_to_generator->port);
            printf("The sequence received was: %i\n",\
                    message_to_generator->sequence);

            printf("//////////////////////////////////////////////END OF "\
                   "GENERATE NEW PACKET////////////////////////////\n");

         }//if(rcvsuccess == IPC_ERROR) {

      }//while (done == PLZCONTINUE) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE GENERATOR: Stopping the probe"\
                                       " generator");
      printf("%s\n",textbuffer);

}//void probe_generator() {
```

PROBE_LOADER.C

```
/*****************************************************************************
 * This is the module that loads the probing engine with its initial data
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_loader.c
 *
 *****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <netinet/in.h>
#include <libnet.h>
#include "probe_main.h"
#include "probe_loader.h"
#include "sys_log.h"


#define TROUBLESHOOT 0          // 1 = load sources  2 = load edges
                                // 3 = load previous stop set
                                // 4 = load previous stop set and turn on print
                                // 5 = load probe list
                                // 6 = load node
                                // 7 = insert node
                                // 8 = load_alias_list
#define BASE 10                 // Base used for conversion numbers for strnl


/*****************************************************************************
 *
 * Define externals and globals
 *
 *****************************************************************************/

   // Set by calls to the library functions to define errors encountered
   extern int errno;

   // Buffer to hold our message for syslog
          char              textbuffer[MAX_MESSAGE];

/*****************************************************************************
 *
 * Subroutines headers
 *
 *****************************************************************************/

  int check_source (struct in6_addr address);

/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/


  int load_sources(void){

     FILE *probe_source_stream;
     // name of the probe source file
     char probe_source_filename[] = "probe_source.txt";
     // Pointer to a string for reading the file
     char *in_string              = NULL;
     // The number of characters we read in
```

```
 int  length                 = ZERO;
 // Counter used in the loop for reading
 int  count                  = ZERO;
 // Value from converting strings to binary IPv6 addr
 int  check                  = ZERO;

snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Starting load of sources");
 call_sys_log(textbuffer);

 /*****************************************************************************
 *
 * Open the file
 *
 *****************************************************************************/

 probe_source_stream = fopen (probe_source_filename, "r");

 if (probe_source_stream == NULL) {

     snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Unable to open " \
                                      "probe_sources.txt load file");
     call_sys_log(textbuffer);
     return -1;

 } else {

    /*****************************************************************************
    *
    * Get the number of structs in the file
    *
    *****************************************************************************/

          in_string           = (char *) malloc (MAX_LINE_LENGTH);
    int   temp                = (MAX_LINE_LENGTH - 1);
    // Used for string to number
    char* endptr              = NULL;

    if (in_string == NULL) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable to " \
                                        "allocate memory for reading");
       call_sys_log(textbuffer);
       fclose(probe_source_stream);
       return -1;

    }//if (in_string == NULL) {

    length = getline(&in_string, &temp, probe_source_stream);
    if (length <= ZERO) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable to" \
                " read the number of sources from probe_source.txt");
       call_sys_log(textbuffer);
       free(in_string);
       fclose(probe_source_stream);
       return -1;

    }//if (length <= ZERO) {

    in_string[(length - 1)] = '\0'; //remove any \n at the end of the line
    number_of_sources       = (int)  strtoul(in_string, &endptr, BASE);

    if (TROUBLESHOOT==1) {

       fprintf(stderr, "We read in the number %s from the sources file\n",\
                        in_string);

    }//if (TROUBLESHOOT==1) {
```

```
if (number_of_sources == ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "probe_sources.txt has incorrect value for number" \
            " of sources");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_source_stream);
   return -1;

}//if (number_of_sources == ZERO) {

if (TROUBLESHOOT==1) {

   fprintf(stderr, "We converted the number to %i " \
                   "number_of_sources\n", number_of_sources);

}//if (TROUBLESHOOT==1) {

/*********************************************************************
*
* Build our list of sources
*
*********************************************************************/
source_list = (struct sources*) malloc(number_of_sources *
                                       sizeof(struct sources));

if (source_list != NULL){

   for (count=ZERO; count < number_of_sources; count++){

      /*****************************************************************
      *
      * Read the source
      *
      *****************************************************************/

      length = getline(&in_string, &temp, probe_source_stream);
      //remove any \n at the end of the line
      in_string[(length - 1)] = '\0';
      if (length <= ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
                  "problem reading the %i address from " \
                  "probe_source.txt", count);
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_source_stream);
         return -1;

      }//if (length <= ZERO) {

      if (TROUBLESHOOT==1) {

         fprintf(stderr, "We read the address %s for count %i\n", \
                         in_string, count);

      }//if (TROUBLESHOOT==1) {

      check = inet_pton(AF_INET6, in_string,
                        &(source_list[count].address));
      if (check <= ZERO){

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
                  "problem converting the %i address from " \
                  "probe_source.txt", count);
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_source_stream);
```

```
   return -1;

}//if (check <= ZERO){

/*****************************************************************
*
* Read the hop count
*
*****************************************************************/

length = getline(&in_string, &temp, probe_source_stream);
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "unable to read hop count from address %i in " \
            "probe_source.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_source_stream);
   return -1;

}//if (length <= ZERO) {

//remove any \n at the end of the line
in_string[(length - 1)]   = '\0';
source_list[count].hop_count  = (int)
                           strtoul(in_string, &endptr, BASE);

if (TROUBLESHOOT==1) {

   fprintf(stderr, "We read the hop count %s for count %i\n", \
                  in_string, count);

}//if (TROUBLESHOOT==1) {

if (source_list[count].hop_count == ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "probe_sources.txt has incorrect value for hop" \
            " count for address %i", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_source_stream);
   return -1;

}//if (hop_count == ZERO) {

/*****************************************************************
*
* Read the h_value
*
*****************************************************************/

length = getline(&in_string, &temp, probe_source_stream);
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "unable to read h_value from address %i in " \
            "probe_source.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_source_stream);
   return -1;

}//if (length <= ZERO) {

//remove any \n at the end of the line
in_string[(length - 1)]   = '\0';
source_list[count].h_value = (int)
```

```
                          strtoul(in_string, &endptr, BASE);

if (TROUBLESHOOT==1) {

   fprintf(stderr, "We read the h_value %s for count %i\n", \
                   in_string, count);

}//if (TROUBLESHOOT==1) {

if (source_list[count].h_value < ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR:" \
            " probe_sources.txt has incorrect value for " \
            "h_value for address %i", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_source_stream);
   return -1;

}//if (h_value < ZERO) {

/*************************************************************
*
* Set the stop and next to probe lists to NULL, they will be
* filled in by another routine
* Sero out the field idenifying the port or id number
*
*************************************************************/

//Set the stop list to nothing
source_list[count].stop_next        = NULL;
//Set the probe list to nothing
source_list[count].probe_next       = NULL;

/*************************************************************
*
* Read the number of values
*
*************************************************************/

length = getline(&in_string, &temp, probe_source_stream);
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "unable to read number_of_values from address " \
            "%i in probe_source.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_source_stream);
   return -1;

}//if (length <= ZERO) {

//remove any \n at the end of the line
in_string[(length - 1)]   = '\0';
source_list[count].number_of_values = (int)
                              strtoul(in_string, &endptr, BASE);

if (TROUBLESHOOT==1) {

   fprintf(stderr, "We read the number_of_values %s for " \
                   "count %i\n", in_string, count);

}//if (TROUBLESHOOT==1) {

if (source_list[count].number_of_values < ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "probe_sources.txt has incorrect value for " \
```

```
                  "number_of_values for address %i", count);
            call_sys_log(textbuffer);
            free(in_string);
            fclose(probe_source_stream);
            return -1;

         }//if (number_of_values < ZERO) {

         int counter = ZERO;
         for (counter=ZERO; counter < MAX_HOPS; counter++) {

            /*************************************************************
            *
            * Read the value of hop length
            *
            *************************************************************/

            length = getline(&in_string, &temp, probe_source_stream);
            if (length <= ZERO) {

               snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
                        "unable to read the hop length % i for address" \
                        " %i in probe_source.txt", counter,count);
               call_sys_log(textbuffer);
               free(in_string);
               fclose(probe_source_stream);
               return -1;

            }//if (length <= ZERO) {

            //remove any \n at the end of the line
            in_string[(length - 1)]   = '\0';
            source_list[count].hop_length[counter] = (int)
                                    strtoul(in_string, &endptr, BASE);

            if (TROUBLESHOOT==1) {

               fprintf(stderr, "We read the hop length %s for count " \
                               "%i\n", in_string, count);

            }//if (TROUBLESHOOT==1) {

            if (source_list[count].hop_length[counter] < ZERO) {

               snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
                        "probe_sources.txt has incorrect value for " \
                        "hop length %i for address %i",(counter), count);
               call_sys_log(textbuffer);
               free(in_string);
               fclose(probe_source_stream);
               return -1;

            }//if (hop_length[counter] < ZERO) {


         }// for (counter=ZERO; counter < MAX_HOPS; counter++) {

      }//for (count=ZERO; count < number_of_sources; count++){

   } else {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: Unable to" \
               " allocate memory to load probe_sources.txt");
      call_sys_log(textbuffer);
      free(in_string);
      fclose(probe_source_stream);
      return -1;

   }//if (source_list != NULL){
```

```
      /*********************************************************************
      *
      * Done, free everything up
      *
      *********************************************************************/
      free(in_string);
      fclose(probe_source_stream);

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Finished load of " \
                                       "sources");
      call_sys_log(textbuffer);
      return ZERO;

   }//if (probe_source_stream == NULL) {


}//int load_sources(void){

int load_edges(void){

   FILE *probe_edges_stream;
   // name of the probe edges file
   char probe_edges_filename[]  = "probe_edges.txt";
   // Pointer to a string for reading the file
   char *in_string              = NULL;
   // The number of characters we read in
   int  length                  = ZERO;
   // Counter used in the loop for reading
   int  count                   = ZERO;
   // Value from converting strings to binary IPv6 addr
   int  check                   = ZERO;

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Starting load of edges ");
   call_sys_log(textbuffer);

   /*********************************************************************
   *
   * Open the edge file
   *
   *********************************************************************/
   probe_edges_stream = fopen (probe_edges_filename, "r");

   if (probe_edges_stream == NULL) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Unable to open " \
                                        "probe_edges.txt load file");
       call_sys_log(textbuffer);
       return -1;

   } else {

      /*********************************************************************
      *
      * This file contains the anonymous node number were working with for
      * use in marking anonymous
      * nodes.  The first number in this file is that number.
      *
      *********************************************************************/

            in_string           = (char *) malloc (MAX_LINE_LENGTH);
      int   temp                = (MAX_LINE_LENGTH - 1);
      // Used for string to number
      char* endptr              = NULL;

      if (in_string == NULL){

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable to" \
                                       " allocate memory for reading file");
```

```
   call_sys_log(textbuffer);
   fclose(probe_edges_stream);
   return -1;

}//if (in_string == NULL){

length  = getline(&in_string, &temp, probe_edges_stream);
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable to" \
                       " anonymous node number from probe_edges.txt");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (length <= ZERO) {

in_string[(length - 1)] = '\0'; //remove any \n at the end of the line
anonymous_address.in6_u.u6_addr32[0] = htonl(0xfe800000);
anonymous_address.in6_u.u6_addr32[1] = 0x00000000;
anonymous_address.in6_u.u6_addr32[2] = 0x00000000;
anonymous_number = (unsigned int)  strtoul(in_string, &endptr, 16);
anonymous_address.in6_u.u6_addr32[3] = htonl(anonymous_number);

if (TROUBLESHOOT==2) {

   fprintf(stderr, "We read in the number %s from the edges file " \
                   "for anonymous number\n", in_string);

}//if (TROUBLESHOOT==2) {


if (anonymous_address.in6_u.u6_addr32[3] <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "probe_edges.txt has incorrect value for " \
            "anonymous number");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (number_of_edges == ZERO) {

if (TROUBLESHOOT==2) {

   fprintf(stderr, "We converted the number to %i anonymous " \
                   "number\n", anonymous_address.in6_u.u6_addr32[3]);

}//if (TROUBLESHOOT==2) {


/*********************************************************************
*
* Read the number of edge structs in the file
*
*********************************************************************/

length = getline(&in_string, &temp, probe_edges_stream);
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable to " \
            "read the number of edges from probe_edges.txt");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;
```

```
}//if (length <= ZERO) {

in_string[(length - 1)] = '\0'; //remove any \n at the end of the line
number_of_edges         = (int)  strtoul(in_string, &endptr, BASE);

if (TROUBLESHOOT==2) {

   fprintf(stderr, "We read in the number %s from the edges file\n",\
                    in_string);

}//if (TROUBLESHOOT==2) {


if (number_of_edges < ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "probe_edges.txt has incorrect value for number of edges");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (number_of_edges == ZERO) {

if (TROUBLESHOOT==2) {

   fprintf(stderr, "We converted the number to %i number_of_edges\n", \
                   number_of_edges);

}//if (TROUBLESHOOT==2) {

/*********************************************************************
*
* Allocate the memory needed to hold all the structs
*
*********************************************************************/

edge_list = (struct edges*) malloc(MAX_EDGES * sizeof(struct edges));

if (edge_list != NULL) {

   for (count=ZERO; count < number_of_edges; count++){

      /*************************************************************
      *
      * Read edge v1
      *
      *************************************************************/

      length = getline(&in_string, &temp, probe_edges_stream);
      //remove any \n at the end of the line
      in_string[(length - 1)] = '\0';
      if (length <= ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
                  "problem reading edge_v1 %i address from " \
                  "probe_edges.txt", count);
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_edges_stream);
         return -1;

      }//if (length <= ZERO) {

      if (TROUBLESHOOT==2) {

         fprintf(stderr, "We read the address %s for count %i\n", \
                         in_string, count);
```

```
}//if (TROUBLESHOOT==2) {

check = inet_pton(AF_INET6, in_string,
                  &(edge_list[count].edge_v1));
if (check <= ZERO){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "problem converting edge_v1 %i address from " \
            "probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (check <= ZERO){

/*****************************************************************
*
* Read edge v2
*
*****************************************************************/

length = getline(&in_string, &temp, probe_edges_stream);
//remove any \n at the end of the line
in_string[(length - 1)] = '\0';
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "problem reading edge_v2 %i address from " \
            "probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (length <= ZERO) {

if (TROUBLESHOOT==2) {

   fprintf(stderr, "We read the address %s for count %i\n",\
                   in_string, count);

}//if (TROUBLESHOOT==2) {

check = inet_pton(AF_INET6, in_string,
                  &(edge_list[count].edge_v2));
if (check <= ZERO){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR:" \
            " problem converting edge_v2 %i address from" \
            " probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (check <= ZERO){

/*****************************************************************
*
* Read edge starting source
*
*****************************************************************/

length  = getline(&in_string, &temp, probe_edges_stream);
//remove any \n at the end of the line
in_string[(length - 1)] = '\0';
if (length <= ZERO) {
```

```
   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "problem reading starting source %i address " \
            "from probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (length <= ZERO) {

if (TROUBLESHOOT==2) {

   fprintf(stderr, "We read the address %s for count %i\n", \
                   in_string, count);

}//if (TROUBLESHOOT==2) {

check = inet_pton(AF_INET6, in_string,
                  &(edge_list[count].starting_source));
if (check <= ZERO){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "problem converting starting source %i address " \
            "from probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (check <= ZERO){

/*****************************************************************
*
* Read edge final dest
*
*****************************************************************/

length = getline(&in_string, &temp, probe_edges_stream);
//remove any \n at the end of the line
in_string[(length - 1)] = '\0';
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "problem reading final dest %i address from " \
            "probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (length <= ZERO) {

if (TROUBLESHOOT==2) {

   fprintf(stderr, "We read the address %s for count %i\n", \
                   in_string, count);

}//if (TROUBLESHOOT==2) {

check = inet_pton(AF_INET6, in_string,
                  &(edge_list[count].final_dest));
if (check <= ZERO){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "problem converting final dest %i address from" \
            " probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
```

```
         return -1;

      }//if (check <= ZERO){

      /*****************************************************************
      *
      * Read hop count
      *
      *****************************************************************/

      length = getline(&in_string, &temp, probe_edges_stream);
      if (length <= ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
                  "unable to read edges hop count for %i in " \
                  "probe_edges.txt", count);
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_edges_stream);
         return -1;

      }//if (length <= ZERO) {

      //remove any \n at the end of the line
      in_string[(length - 1)]   = '\0';
      edge_list[count].hop_count  = (int)
                                 strtoul(in_string, &endptr, BASE);

      if (TROUBLESHOOT==2) {

         fprintf(stderr, "We read the hop count %s for count " \
                         " %i\n", in_string, count);

      }//if (TROUBLESHOOT==2) {

      if (edge_list[count].hop_count < ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR:" \
                  " probe_edges.txt has incorrect value for hop" \
                  " count for %i", count);
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_edges_stream);
         return -1;

      }//if (number_of_edges == ZERO) {

   }//for (count=ZERO; count < number_of_edges; count++){


} else {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable to " \
                                 "obtain memory for probe_edges.txt");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (edge != NULL) {

free(in_string);
fclose(probe_edges_stream);

if (TROUBLESHOOT==2) {

   for (count=ZERO; count < number_of_edges; count++){

      fprintf(stderr, "The hop count for edge %i is: %i\n",count, \
```

```
                           edge_list[count].hop_count);

         }//for (count=ZERO; count < number_of_edges; count++){

      }//if (TROUBLESHOOT==2) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Finished load of " \
                                       "edges ");
      call_sys_log(textbuffer);
      return ZERO;

   }//if (probe_edges_stream == NULL) {

}//int load_edges(void){


int load_stop_list(void){

   /*********************************************************************
   *
   * Note: Except for troubleshooting the stop list is initialized to zero
   *       The rationale is that we may have been in the middle of probing
   *       and don't want the stop
   *       list to make us miss something so for startup a little
   *       redundancy is sacrificed.
   *
   *********************************************************************/

   if (TROUBLESHOOT==4 || TROUBLESHOOT == 3) {

      FILE *probe_stop_stream;
      // name of the probe stop file
      char probe_stop_filename[]    = "probe_stop.txt";
      // Pointer to a string for reading the file
      char *in_string              = NULL;
      // The number of characters we read in
      int  length                  = ZERO;
      // Counter used in the loop for reading
      int  count                   = ZERO;
      // Value from converting strings to binary IPv6 addr
      int  check                   = ZERO;

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Starting load of " \
                                       "stop list ");
      call_sys_log(textbuffer);

      /*********************************************************************
      *
      * Open the stop file
      *
      *********************************************************************/
      probe_stop_stream = fopen (probe_stop_filename, "r");

      if (probe_stop_stream == NULL) {

          snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Unable to open " \
                                           "probe_stop.txt load file");
          call_sys_log(textbuffer);
          return -1;

      } else {

         /*********************************************************************
         *
         * Read the number of global stop structs in the file
         *
         *********************************************************************/

               in_string           = (char *) malloc (MAX_LINE_LENGTH);
```

```
int   temp                = (MAX_LINE_LENGTH - 1);
// Used for string to number
char* endptr              = NULL;

length = getline(&in_string, &temp, probe_stop_stream);
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable" \
            " to read the number of global stops from " \
            "probe_stop.txt");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_stop_stream);
   return -1;

}//if (length <= ZERO) {

//remove any \n at the end of the line
in_string[(length - 1)] = '\0';
number_of_global_stops   = (int)  strtoul(in_string, &endptr, BASE);

if (TROUBLESHOOT==4) {

   fprintf(stderr, "We read in the global stops number %s from " \
                   "the stop file\n", in_string);

}//if (TROUBLESHOOT==4) {


if (number_of_global_stops < ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "probe_stop.txt has incorrect value for number " \
            "of global stops");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_stop_stream);
   return -1;

}//if (number_of_global_stops == ZERO) {

if (TROUBLESHOOT==4) {

   fprintf(stderr, "We converted the number to %i " \
           "number_of_global_stops\n", number_of_global_stops);

}//if (TROUBLESHOOT==4) {

/*****************************************************************
*
* Allocate the memory needed to hold all the structs
* We allocate a maximum of of MAX_EDGES
*
*****************************************************************/

global_stop_list = (struct global_stops*)
                   malloc(MAX_EDGES * sizeof(struct global_stops));

if (global_stop_list == NULL) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: problem" \
            " reading v1 %i address from probe_stop.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_stop_stream);
   return -1;

}//if (global_stop_list == NULL) {
```

```
for (count=ZERO; count < number_of_global_stops; count++){

   /*****************************************************************
   *
   * Read v1
   *
   *****************************************************************/

   length = getline(&in_string, &temp, probe_stop_stream);
   //remove any \n at the end of the line
   in_string[(length - 1)] = '\0';
   if (length <= ZERO) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
               "problem reading v1 %i address from " \
               "probe_stop.txt", count);
      call_sys_log(textbuffer);
      free(in_string);
      fclose(probe_stop_stream);
      return -1;

   }//if (length <= ZERO) {

   if (TROUBLESHOOT==4) {

      fprintf(stderr, "We read the address %s for count %i\n", \
                      in_string, count);

   }//if (TROUBLESHOOT==4) {

   check = inet_pton(AF_INET6, in_string,
                     &(global_stop_list[count].address));
   if (check <= ZERO){

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
               "problem converting v1 %i address from " \
               "probe_stop.txt", count);
      call_sys_log(textbuffer);
      free(in_string);
      fclose(probe_stop_stream);
      return -1;

   }//if (check <= ZERO){

   /*****************************************************************
   *
   * Read destination
   *
   *****************************************************************/

   length = getline(&in_string, &temp, probe_stop_stream);
   //remove any \n at the end of the line
   in_string[(length - 1)] = '\0';
   if (length <= ZERO) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
               "problem reading v2 %i address from " \
               "probe_stop.txt", count);
      call_sys_log(textbuffer);
      free(in_string);
      fclose(probe_stop_stream);
      return -1;

   }//if (length <= ZERO) {

   if (TROUBLESHOOT==4) {

      fprintf(stderr, "We read the address %s for count %i\n", \
                      in_string, count);
```

```
   }//if (TROUBLESHOOT==4) {

   check = inet_pton(AF_INET6, in_string,
                     &(global_stop_list[count].dest_address));
   if (check <= ZERO){

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
               "problem converting v2 %i address from " \
               "probe_stop.txt", count);
      call_sys_log(textbuffer);
      free(in_string);
      fclose(probe_stop_stream);
      return -1;

   }//if (check <= ZERO){

}//for (count=ZERO; count < number_of_sources; count++){


int probe_source = ZERO;
for(probe_source=ZERO;probe_source < number_of_sources;
    probe_source++){

  /*****************************************************************
   *
   * Read the number of stop structs for this source
   *
   *****************************************************************/

   length = getline(&in_string, &temp, probe_stop_stream);
   if (length <= ZERO) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
               "unable to read the number of stops from " \
               "probe_stop.txt for source %i", probe_source);
      call_sys_log(textbuffer);
      free(in_string);
      fclose(probe_stop_stream);
      return -1;

   }//if (length <= ZERO) {

   //remove any \n at the end of the line
   in_string[(length - 1)] = '\0';
   int number_of_stops   = (int)  strtoul(in_string, &endptr, BASE);

   if (TROUBLESHOOT==4) {

      fprintf(stderr, "We read in the stops number %s from the " \
              "stop file for source %i\n", in_string, probe_source);

   }//if (TROUBLESHOOT==4) {


   if (number_of_stops < ZERO) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
               "probe_stop.txt has incorrect value for number" \
               " of stops for source %i", probe_source);
      call_sys_log(textbuffer);
      free(in_string);
      fclose(probe_stop_stream);
      return -1;

   }//if (number_of_global_stops == ZERO) {

   if (TROUBLESHOOT==4) {
```

```
   fprintf(stderr, "We converted the number to %i " \
           "number_of_stops for source %i\n", \
           number_of_stops, probe_source);

}//if (TROUBLESHOOT==4) {

if (number_of_stops > ZERO){

   struct stops *temp_stop_pointer;
   source_list[probe_source].stop_next =
              (struct stops*) malloc(sizeof(struct stops));

   if (source_list[probe_source].stop_next == NULL) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: " \
               "ERROR: Unable to allocate memory to " \
               "load local stop set for source %i", \
               probe_source);
      call_sys_log(textbuffer);
      free(in_string);
      fclose(probe_stop_stream);
      return -1;

   }//if (source_list[count].next == NULL) {

   temp_stop_pointer = source_list[probe_source].stop_next;

   for(count=ZERO;count<number_of_stops;count++){

      /*********************************************************
      *
      * Read stop address
      *
      *********************************************************/

      length = getline(&in_string, &temp, probe_stop_stream);
      //remove any \n at the end of the line
      in_string[(length - 1)] = '\0';
      if (length <= ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR:" \
                  " problem reading stop %i address from " \
                  "probe_stop.txt for source %i", \
                  count, probe_source);
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_stop_stream);
         return -1;

      }//if (length <= ZERO) {

      if (TROUBLESHOOT==4) {

         fprintf(stderr, "We read the address %s for count %i " \
                  "for source %i\n", in_string, count, \
                  probe_source);

      }//if (TROUBLESHOOT==4) {

      check = inet_pton(AF_INET6, in_string,
                        &(temp_stop_pointer->address));
      if (check <= ZERO){

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR:" \
                  " problem converting stop %i address for " \
                  "source %i from probe_stop.txt", \
                  count, probe_source);
         call_sys_log(textbuffer);
         free(in_string);
```

```
                  fclose(probe_stop_stream);
                  return -1;

               }//if (check <= ZERO){

               if (count >= (number_of_stops - 1)) {

                  temp_stop_pointer->next = NULL;

               } else {

                  temp_stop_pointer->next = (struct stops*) \
                                       malloc(sizeof(struct stops));

                  if (temp_stop_pointer->next == NULL) {

                      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER:" \
                              " ERROR: Unable to allocate memory to " \
                              "load local stop set for source %i", \
                              probe_source);
                      call_sys_log(textbuffer);
                      free(in_string);
                      fclose(probe_stop_stream);
                      return -1;

                  } else {

                     temp_stop_pointer = temp_stop_pointer->next;

                  }//if (temp_stop_pointer->next == NULL) {

               }//if (count >= (number_of_stops - 1)) {


            }//for(count=ZERO;count<number_of_stops;count++){

         }//if (number_of_stops > ZERO){

      }//for(probe_source=ZERO;probe_source < number_of_sources; probe_sou


      free(in_string);
      fclose(probe_stop_stream);

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Finished load of" \
                                  " stops ");
      call_sys_log(textbuffer);
      return ZERO;

   }//if (probe_stop_stream == NULL) {

} else {

   number_of_global_stops = ZERO;
   /*********************************************************************
   *
   * Allocate the memory needed to hold all the structs
   * We allocate a maximum of of MAX_EDGES
   *
   *********************************************************************/

   global_stop_list       = (struct global_stops*) \
                            malloc(MAX_EDGES * sizeof(struct global_stops));
   int probe_source = ZERO;
   for(probe_source=ZERO;probe_source<number_of_sources; probe_source++){

      source_list[probe_source].stop_next = NULL;

   }//for(probe_source=ZERO;probe_source < number_of_sources; probe_source
```

```
      return ZERO;

   }//if (TROUBLESHOOT==4 || TROUBLESHOOT == 3) {

}//int load_stop_list(void){

int load_probe_list(void){

   FILE *probe_list_stream;
   // name of the probe list file
   char probe_list_filename[]    = "probe_list.txt";
   // Pointer to a string for reading the file
   char *in_string              = NULL;
   // The number of characters we read in
   int  length                  = ZERO;
   // Counter used in the loop for reading
   int  count                   = ZERO;
   // Value from converting strings to binary IPv6 addr
   int  check                   = ZERO;

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Starting load of probe" \
                                  " list ");
   call_sys_log(textbuffer);

   /*********************************************************************
   *
   * Open the probe list file
   *
   *********************************************************************/
   probe_list_stream = fopen (probe_list_filename, "r");

   if (probe_list_stream == NULL) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Unable to open " \
                                        "probe_list.txt load file");
       call_sys_log(textbuffer);
       return -1;

   } else {


      /*********************************************************************
      *
      * Read the number of items in the each sources lists
      *
      *********************************************************************/

            in_string           = (char *) malloc (MAX_LINE_LENGTH);
      int   temp                = (MAX_LINE_LENGTH - 1);
      // Used for string to number
      char* endptr              = NULL;

      int probe_source = ZERO;
      for(probe_source=ZERO;probe_source<number_of_sources; probe_source++){

         length = getline(&in_string, &temp, probe_list_stream);
         if (length <= ZERO) {

            snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable " \
                     "to read the number of items from probe_list.txt for " \
                     "source %i", probe_source);
            call_sys_log(textbuffer);
            free(in_string);
            fclose(probe_list_stream);
            return -1;

         }//if (length <= ZERO) {

         //remove any \n at the end of the line
```

```
in_string[(length - 1)] = '\0';
int number_of_items   = (int)  strtoul(in_string, &endptr, BASE);

if (TROUBLESHOOT==5) {

   fprintf(stderr, "We read in the number %s from the list file" \
                   " for source %i\n", in_string, probe_source);

}//if (TROUBLESHOOT==5) {

source_list[probe_source].remaining_probe_counter = number_of_items;

if (number_of_items < ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "probe_list.txt has incorrect value for number" \
            " of probes for source %i", probe_source);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_list_stream);
   return -1;

}//if (number_of_global_stops == ZERO) {

if (TROUBLESHOOT==5) {

   fprintf(stderr, "We converted the number to %i number_of_items" \
                   " for source %i\n", number_of_items, \
                   probe_source);

}//if (TROUBLESHOOT==5) {

if (number_of_items > ZERO){

   struct remaining_probes *temp_probe_pointer;
   source_list[probe_source].probe_next =
                        (struct remaining_probes*)
                         malloc(sizeof(struct remaining_probes));

   if (source_list[probe_source].probe_next == NULL) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
               "Unable to allocate memory to load probe list " \
               "for source %i", probe_source);
      call_sys_log(textbuffer);
      free(in_string);
      fclose(probe_list_stream);
      return -1;

   }//if (source_list[count].probe_next == NULL) {

   temp_probe_pointer = source_list[probe_source].probe_next;

   for(count=ZERO;count<number_of_items;count++){

      length                  = getline(&in_string, &temp,
                                        probe_list_stream);
      //remove any \n at the end of the line
      in_string[(length - 1)] = '\0';
      if (length <= ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR:" \
                  " problem reading probe %i address from " \
                  "probe_list.txt for source %i", \
                  count, probe_source);
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_list_stream);
         return -1;
```

```
            }//if (length <= ZERO) {

            if (TROUBLESHOOT==5) {

               fprintf(stderr, "We read the address %s for count %i for" \
                               " source %i\n", in_string, \
                               count, probe_source);

            }//if (TROUBLESHOOT==5) {

            check = inet_pton(AF_INET6, in_string,
                              &(temp_probe_pointer->address));
            if (check <= ZERO){

               snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
                        "problem converting probe %i address for source" \
                        " %i from probe_list.txt", count, probe_source);
               call_sys_log(textbuffer);
               free(in_string);
               fclose(probe_list_stream);
               return -1;

            }//if (check <= ZERO){

            if (count >= (number_of_items - 1)) {

               temp_probe_pointer->next = NULL;

            } else {

               temp_probe_pointer->next = (struct remaining_probes*)
                                malloc(sizeof(struct remaining_probes));

               if (temp_probe_pointer->next == NULL) {

                  snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: " \
                           "ERROR: Unable to allocate memory to " \
                           "load probe set for source %i", probe_source);
                  call_sys_log(textbuffer);
                  free(in_string);
                  fclose(probe_list_stream);
                  return -1;

                } else {

                    temp_probe_pointer = temp_probe_pointer->next;

                }//if (temp_probe_pointer->next == NULL) {

            }//if (count >= (number_of_stops - 1)) {

         }//for(count=ZERO;count<number_of_iems;count++){

      }//if (number_of_items > ZERO){

   }//for(probe_source=ZERO;probe_source < number_of_sources; probe_source


   free(in_string);
   fclose(probe_list_stream);

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Finished load of " \
                                    "probes");
   call_sys_log(textbuffer);
   return ZERO;

}//if (probe_list_stream == NULL) {
```

```
}//int load_probe_list(void){

int load_nodes(void){

   FILE *probe_edges_stream;
   // name of the probe edges file
   char probe_edges_filename[]  = "probe_edges.txt";
   // Pointer to a string for reading the file
   char *in_string              = NULL;
   // The number of characters we read in
   int  length                  = ZERO;
   // Counter used in the loop for reading
   int  count                   = ZERO;
   // Value from converting strings to binary IPv6 addr
   int  check                   = ZERO;

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Starting load of nodes" \
                                 " from edge file ");
   call_sys_log(textbuffer);

   /*************************************************************************
   *
   * Open the edge file
   *
   *************************************************************************/
   probe_edges_stream = fopen (probe_edges_filename, "r");

   if (probe_edges_stream == NULL) {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Unable to open " \
                                      "probe_edges.txt load file");
       call_sys_log(textbuffer);
       return -1;

   } else {

      /************************************************************************
      *
      * This file contains the anonymous node number were working with for
      * use in marking anonymous
      * nodes.  The first number in this file is that number.
      *
      ************************************************************************/

            in_string           = (char *) malloc (MAX_LINE_LENGTH);
      int   temp                = (MAX_LINE_LENGTH - 1);
      // Used for string to number
      char* endptr              = NULL;

      if (in_string == NULL){

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable to" \
                  " allocate memory for reading file");
         call_sys_log(textbuffer);
         fclose(probe_edges_stream);
         return -1;

      }//if (in_string == NULL){

      length = getline(&in_string, &temp, probe_edges_stream);
      if (length <= ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable to" \
                  " anonymous node number from probe_edges.txt");
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_edges_stream);
         return -1;
```

```
}//if (length <= ZERO) {

in_string[(length - 1)] = '\0'; //remove any \n at the end of the line
anonymous_address.in6_u.u6_addr32[0] = htonl(0xfe800000);
anonymous_address.in6_u.u6_addr32[1] = 0x00000000;
anonymous_address.in6_u.u6_addr32[2] = 0x00000000;
anonymous_number = (unsigned int)  strtoul(in_string, &endptr, 16);
anonymous_address.in6_u.u6_addr32[3] = htonl(anonymous_number);

if (TROUBLESHOOT==6) {

   fprintf(stderr, "We read in the number %s from the edges file for" \
                   " anonymous number\n", in_string);

}//if (TROUBLESHOOT==6) {


if (anonymous_address.in6_u.u6_addr32[3] <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
           "probe_edges.txt has incorrect value for anonymous number");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (number_of_edges == ZERO) {

if (TROUBLESHOOT==6) {

   fprintf(stderr, "We converted the number to %i anonymous number\n",\
                    anonymous_address.in6_u.u6_addr32[3]);

}//if (TROUBLESHOOT==6) {


/*********************************************************************
*
* Read the number of edge structs in the file
*
*********************************************************************/

length                  = getline(&in_string, &temp, probe_edges_stream);
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable to " \
            "read the number of edges from probe_edges.txt");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (length <= ZERO) {

in_string[(length - 1)] = '\0'; //remove any \n at the end of the line
number_of_edges         = (int)  strtoul(in_string, &endptr, BASE);

if (TROUBLESHOOT==6) {

   fprintf(stderr, "We read in the number %s from the edges file\n",\
                    in_string);

}//if (TROUBLESHOOT==6) {

if (number_of_edges < ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "probe_edges.txt has incorrect value for number of edges");
   call_sys_log(textbuffer);
```

```
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (number_of_edges == ZERO) {

if (TROUBLESHOOT==6) {

   fprintf(stderr, "We converted the number to %i number_of_edges\n",\
                   number_of_edges);

}//if (TROUBLESHOOT==6) {

/*********************************************************************
*
* Allocate the memory needed to hold all the node structs
*
*********************************************************************/

struct edges          *current_edge;
current_edge = (struct edges*) malloc(sizeof(struct edges));

alias_list   = (struct alias_nodes*) malloc(number_of_edges * 2 *
                sizeof(struct edges));

/*********************************************************************
*
* Get our position in the file so we can come back here
*
*********************************************************************/

if ((current_edge != NULL) && (alias_list != NULL)) {

   for (count=ZERO; count < number_of_edges; count++){

      /*****************************************************************
      *
      * Read edge v1
      *
      *****************************************************************/

      length = getline(&in_string, &temp, probe_edges_stream);
      //remove any \n at the end of the line
      in_string[(length - 1)] = '\0';
      if (length <= ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
                  "problem reading edge_v1 %i address from " \
                  "probe_edges.txt", count);
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_edges_stream);
         return -1;

      }//if (length <= ZERO) {

      if (TROUBLESHOOT==6) {

         fprintf(stderr, "We read the address %s for count %i\n", \
                         in_string, count);

      }//if (TROUBLESHOOT==6) {

      struct in6_addr temp_address;
      check = inet_pton(AF_INET6, in_string, &temp_address);
      if (check <= ZERO){

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR:" \
                  " problem converting edge_v1 %i address from" \
```

```
            " probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (check <= ZERO){

/*************************************************************
*
* Make sure the address is not a source address or an anonymous
* address
*
*************************************************************/

if (cmpaddr(&temp_address, &empty, 16) != ZERO) {

   if (check_source (temp_address) == NOT_FOUND) {

      int temp = insert_node(temp_address);

   }//if (check_source (temp_address) == NOT_FOUND) {

}//if (cmpaddr(&temp_address, &empty, 16) != ZERO) {

/*************************************************************
*
* Read edge v2
*
*************************************************************/

length = getline(&in_string, &temp, probe_edges_stream);
//remove any \n at the end of the line
in_string[(length - 1)] = '\0';
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "problem reading edge_v2 %i address from " \
            "probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (length <= ZERO) {

if (TROUBLESHOOT==6) {

   fprintf(stderr, "We read the address %s for count %i\n", \
                   in_string, count);

}//if (TROUBLESHOOT==6) {

check = inet_pton(AF_INET6, in_string, &temp_address);
if (check <= ZERO){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "problem converting edge_v1 %i address from " \
            "probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (check <= ZERO){

/*************************************************************
*
* Make sure the address is not a source address or an anonymous
```

```
* address
*
*************************************************************/

if (cmpaddr(&temp_address, &empty, 16) != ZERO) {

   if (check_source (temp_address) == NOT_FOUND) {

      int temp = insert_node(temp_address);

   }//if (check_source (temp_address) == NOT_FOUND) {

}//if (cmpaddr(&temp_address, &empty, 16) != ZERO) {


/*************************************************************
*
* Read edge starting source and discard
*
*************************************************************/

length = getline(&in_string, &temp, probe_edges_stream);
//remove any \n at the end of the line
in_string[(length - 1)] = '\0';
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "problem reading starting source %i address " \
            "from probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (length <= ZERO) {

if (TROUBLESHOOT==6) {

   fprintf(stderr, "We read the address %s for count %i\n", \
                  in_string, count);

}//if (TROUBLESHOOT==6) {

/*************************************************************
*
* Read edge final dest and discard
*
*************************************************************/

length = getline(&in_string, &temp, probe_edges_stream);
//remove any \n at the end of the line
in_string[(length - 1)] = '\0';
if (length <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
            "problem reading final dest %i address from " \
            "probe_edges.txt", count);
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if (length <= ZERO) {

if (TROUBLESHOOT==6) {

   fprintf(stderr, "We read the address %s for count %i\n", \
                   in_string, count);
```

```
      }//if (TROUBLESHOOT==6) {

      /*************************************************************
      *
      * Read hop count and discard
      *
      *************************************************************/

      length = getline(&in_string, &temp, probe_edges_stream);
      if (length <= ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: " \
                  "unable to read edges hop count for %i in " \
                  "probe_edges.txt", count);
         call_sys_log(textbuffer);
         free(in_string);
         fclose(probe_edges_stream);
         return -1;

      }//if (length <= ZERO) {

      if (TROUBLESHOOT==6) {

         fprintf(stderr, "We read the hop count %s for count %i\n", \
                        in_string, count);

      }//if (TROUBLESHOOT==6) {

   }//for (count=ZERO; count < number_of_sources; count++){


} else {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: ERROR: unable " \
            "to obtain memory for probe_edges.txt");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(probe_edges_stream);
   return -1;

}//if ((current_edge != NULL) && (alias_list != NULL)) {

free(in_string);
fclose(probe_edges_stream);

snprintf(textbuffer, MAX_MESSAGE,"PROBE LOADER: Finished load of" \
                                 " nodes");
call_sys_log(textbuffer);
if (TROUBLESHOOT == 6) {

   // Pointer to a string for writing to the file
   char                    *out_string                   = NULL;
   out_string          = (char *) malloc (MAX_LINE_LENGTH);

   if (out_string == NULL){

      snprintf(textbuffer, MAX_MESSAGE,"PROBER LOADER: ERROR: " \
               "unable to allocate memory for strings");
      call_sys_log(textbuffer);
      return -1;

   }//if (in_string == NULL){

   fprintf(stderr,"The node list is the following\n");
   fprintf (stderr, "The number of nodes is %i\n", number_of_nodes);
   for(count=ZERO;count<number_of_nodes;count++) {

       if (inet_ntop(AF_INET6, &(alias_list[count].original_address),
                     out_string, INET6_ADDRSTRLEN) == NULL){
```

```
                  snprintf(textbuffer, MAX_MESSAGE,"ALIAS SAVE: ERROR: " \
                           "problem converting original %i address for " \
                           "alias.txt", count);
                  call_sys_log(textbuffer);
                  free(out_string);
                  return -1;

               }//if (inet_ntop(AF_INET6, &(alias_list[count].original_address) ,
out_string, INET6_ADDRSTRLEN) == NULL){

               fprintf (stderr, "%s     ", out_string);

               if (inet_ntop(AF_INET6, &(alias_list[count].resolved_address) ,
out_string, INET6_ADDRSTRLEN) == NULL){

                  snprintf(textbuffer, MAX_MESSAGE,"ALIAS SAVE: ERROR: problem converting
original %i address for alias.txt", (count + 1));
                  call_sys_log(textbuffer);
                  free(out_string);
                  return -1;

               }//if (inet_ntop(AF_INET6, &(alias_list[count].original_address) ,
out_string, INET6_ADDRSTRLEN) == NULL){

               fprintf (stderr, "%s\n", out_string);

            }//for(count=ZERO;count<number_of_nodes;count++) {

         }//if (TROUBLESHOOT == 6) {

         return ZERO;

      }//if (probe_edges_stream == NULL) {

   }//int load_nodes(void){


   int insert_node(struct in6_addr address) {

      int middle = ZERO;  // Index into the global array to locate the item
      int bottom = ZERO;
      int result = ZERO;  // The result of a comparison
      int count  = ZERO;  // Counter for loops
      int top    = (number_of_nodes  - 1);

      if (TROUBLESHOOT==7) {

         fprintf(stderr, "We are starting insert node\n");

      }//if (TROUBLESHOOT==7) {

      while (bottom <= top) {

         middle = (bottom + top) / 2;
         result = cmpaddr(&address, &(alias_list[middle].original_address), 128);
         if (TROUBLESHOOT==7) {

           fprintf(stderr, "The value of bottom = %i\n", bottom);
           fprintf(stderr, "The value of top    = %i\n", top);
           fprintf(stderr, "The value of middle = %i\n", middle);
           fprintf(stderr, "The value of result = %i\n", result);

         }//if (TROUBLESHOOT==7) {

         if (result == ZERO){

            /*****************************************************************
            *
```

```
            * If we are here we found the node
            *
            ********************************************************************/

            return FOUND;

         } else if (result < ZERO) {

            top = middle - 1;

         } else {

            bottom = middle + 1;

         }//if (result == ZERO){

      }//while (bottom <= top) {


/*************************************************************************************
******
      *
      * If we are here we did not find the node and need to insert it
      *

*************************************************************************************
*****/

      if (bottom > number_of_nodes) {

         alias_list[number_of_nodes].original_address      = address;
         alias_list[number_of_nodes].resolved_address      = empty;
         number_of_nodes++;
         return NOT_FOUND_AND_INSERTED;

      } else {

         for(count=number_of_nodes;count>bottom;count--) {

            alias_list[count].original_address      = alias_list[(count-
1)].original_address;
            alias_list[count].resolved_address      = alias_list[(count-
1)].resolved_address;

         }//for(count=number_of_nodes;count>bottom;count--) {
         alias_list[bottom].original_address      = address;
         alias_list[bottom].resolved_address      = empty;
         number_of_nodes++;
         return NOT_FOUND_AND_INSERTED;

      }//if (bottom > number_of_nodes) {

      if (TROUBLESHOOT==7) {

         fprintf(stderr, "We are ending insert node\n");

      }//if (TROUBLESHOOT==7) {

   }//void insert_node(struct in6_addr address) {

   int check_source (struct in6_addr address) {

      int count = ZERO; // counter for the loop

      for(count=ZERO; count<number_of_sources;count++) {

         if(cmpaddr(&address, &(source_list[count].address), 128) == ZERO) {

            return FOUND;
```

```
      }//if(cmpaddr(&address, &(source_list[count].address), 128) == ZERO) {

   }//for(count=ZERO; count<number_of_sources;count++) {

   return NOT_FOUND;

}//int check_source (struct in6_addr address)

 int load_alias_list(void) {

    char                    *in_string                  = NULL;
    // Counter used in the loop for reading
    int                     count                       = ZERO;
    // Value from converting strings to binary IPv6 addr
    int                     check                       = ZERO;
    int                     length                      = ZERO;
    FILE                    *alias_stream;
    // name of the alias file
    char                    alias_filename[]            = "alias.txt";

    snprintf(textbuffer, MAX_MESSAGE,"LOAD ALIAS: Starting the load of "\
                                  "alias file");
    call_sys_log(textbuffer);

    /*************************************************************************
    *
    * Create the space for the ourstring
    *
    *************************************************************************/

    in_string                = (char *) malloc (MAX_LINE_LENGTH);
    int   temp               = (MAX_LINE_LENGTH - 1);
    // Used for string to number
    char* endptr             = NULL;

    if (in_string == NULL){

       snprintf(textbuffer, MAX_MESSAGE,"LOAD ALIAS: ERROR: unable to "\
                                     "allocate memory for strings");
       call_sys_log(textbuffer);
       return -1;

    }//if (in_string == NULL){

    /*************************************************************************
    *
    * Open the alias file
    *
    *************************************************************************/
    alias_stream = fopen (alias_filename, "r");

    if (alias_stream == NULL) {

        snprintf(textbuffer, MAX_MESSAGE,"ALIAS LOADER: Unable to open "\
                                      "alias.txt file");
        call_sys_log(textbuffer);
        return -1;

    } else {

      /*************************************************************************
      *
      * Read the number of aliases
      *
      *************************************************************************/

      length                 = getline(&in_string, &temp, alias_stream);
      if (length <= ZERO) {
```

```
   snprintf(textbuffer, MAX_MESSAGE,"ALIAS LOADER: ERROR: unable to " \
            "read the number of nodes from alias.txt");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(alias_stream);
   return -1;

}//if (length <= ZERO) {

in_string[(length - 1)] = '\0'; //remove any \n at the end of the line
number_of_nodes         = (int)  strtoul(in_string, &endptr, BASE);

if (TROUBLESHOOT==8) {

   fprintf(stderr, "We read in the number %s from the alias file\n", \
                  in_string);

}//if (TROUBLESHOOT==8) {

if (number_of_nodes < ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"ALIAS LOADER: ERROR: alias.txt " \
            "has incorrect value for number of nodes");
   call_sys_log(textbuffer);
   free(in_string);
   fclose(alias_stream);
   return -1;

}//if (number_of_nodes < ZERO) {

if (TROUBLESHOOT==8) {

   fprintf(stderr, "We converted the number to %i number_of_nodes\n",\
                   number_of_nodes);

}//if (TROUBLESHOOT==8) {

/*********************************************************************
*
* Write the aliases
*
*********************************************************************/

for (count=ZERO; count < number_of_nodes; count++){

    /*****************************************************************
    *
    * Read original address
    *
    *****************************************************************/

    length                  = getline(&in_string, &temp, alias_stream);
    //remove any \n at the end of the line
    in_string[(length - 1)] = '\0';
    if (length <= ZERO) {

       snprintf(textbuffer, MAX_MESSAGE,"ALIAS LOADER: ERROR: " \
                "problem reading original %i address from alias.txt",\
                 count);
       call_sys_log(textbuffer);
       free(in_string);
       fclose(alias_stream);
       return -1;

    }//if (length <= ZERO) {

    if (TROUBLESHOOT==8) {
```

```
            fprintf(stderr, "We read the address %s for count %i\n", \
                            in_string, count);

        }//if (TROUBLESHOOT==8) {

        check = inet_pton(AF_INET6, in_string,
                          &(alias_list[count].original_address));
        if (check <= ZERO){

            snprintf(textbuffer, MAX_MESSAGE,"ALIAS LOADER: ERROR: " \
                     "problem converting original %i address from " \
                     "alias.txt", count);
            call_sys_log(textbuffer);
            free(in_string);
            fclose(alias_stream);
            return -1;

        }//if (check <= ZERO){

        /*****************************************************************
        *
        * Read resolved address
        *
        *****************************************************************/

        length                  = getline(&in_string, &temp, alias_stream);
        //remove any \n at the end of the line
        in_string[(length - 1)] = '\0';
        if (length <= ZERO) {

            snprintf(textbuffer, MAX_MESSAGE,"ALIAS LOADER: ERROR: problem"\
                     " reading resolved %i address from alias.txt", count);
            call_sys_log(textbuffer);
            free(in_string);
            fclose(alias_stream);
            return -1;

        }//if (length <= ZERO) {

        if (TROUBLESHOOT==8) {

            fprintf(stderr, "We read the address %s for count %i\n", \
                            in_string, count);

        }//if (TROUBLESHOOT==8) {

        check = inet_pton(AF_INET6, in_string,
                          &(alias_list[count].resolved_address));
        if (check <= ZERO){

            snprintf(textbuffer, MAX_MESSAGE,"ALIAS LOADER: ERROR: problem"\
                     " converting resolved %i address from alias.txt", \
                     count);
            call_sys_log(textbuffer);
            free(in_string);
            fclose(alias_stream);
            return -1;

        }//if (check <= ZERO){

    }//for (count=ZERO; count < number_of_nodes; count++){

    free(in_string);
    fclose(alias_stream);

    snprintf(textbuffer, MAX_MESSAGE,"ALIAS LOADER: Finished loading "\
                                     "addresses ");
    call_sys_log(textbuffer);
    return ZERO;
```

```
   }//if (alias_stream == NULL) {
}//int load_alias_list(void) {
```

PROBE_LOADER.H

```
/*****************************************************************************
 * This is the header file for the program that initially loads the program
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_loader.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_PROBE_LOADER
#define INCLUSION_GUARD_PROGRAM_PROBE_LOADER

   int load_sources(void);
   int load_edges(void);
   int load_stop_list(void);
   int load_probe_list(void);
   int load_nodes(void);
   int load_alias_list(void);

#endif //INCLUSION_GUARD_PROGRAM_PROBE_LOADER
```

PROBE_MAIN.C

```
/*****************************************************************************
* This is the main program of the probing engine and controls all operation
*
*
* Author: Robert J. Poulin, Capt, USAF
*
* Program name: probe_main.c
*
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <netinet/in.h>
#include <libnet.h>
#include <signal.h>
#include <unistd.h>
#include <wait.h>
#include "probe_main.h"
#include "probe_generator.h"
#include "probe_receive.h"
#include "probe_utils.h"
#include "probe_loader.h"
#include "probe_ckpnt.h"
#include "probe_stop.h"
#include "sys_log.h"
#include "probe_main_helper.h"
#include <sys/socket.h>
#include <arpa/inet.h>
#include <pcap.h>


#define _GNU_SOURCE
#define TROUBLESHOOT 0    // 1  - startup in main only
                          // 99 - We are troubleshooting other routines do
                          // not stop if they die

/*****************************************************************************
*
* Define constants
*
*****************************************************************************/

   // The messaging function calls -1 an error
   const  int               IPC_ERROR        = -1;
   // The messaging function calls -1 an error
   const  int               LIBNET_ERROR     = -1;
   // Flag stating we are going in the positive direction
   const  int               positive         =  1;
   // Flag stating we are going in the negative direction
   const  int               negative         = -1;


/*****************************************************************************
*
* Define externals and globals
*
*****************************************************************************/

   // Set by calls to the library functions to define errors encountered
   extern int errno;

   // Buffer for messages to the generator function
   struct generator_buffer  *message_to_generator;
   // Buffer for messages from receive  function
```

```
   struct receive_buffer    *message_from_receiver;
   // Pointer to the list of sources
   struct sources           *source_list;
   // Pointer to the list of edges
   struct edges             *edge_list;
   // Pointer to the global stop set
   struct global_stops      *global_stop_list;
   // Pointer to the local stop set
          void*             *probe_source_stop_list;
   // Pointer to the alias list for alias resolution
   struct alias_nodes       *alias_list;

   struct messagebuffer     sendbuffer, recvbuffer;
   // Flag address stating there is no address
   struct in6_addr          empty;
   // address used to mark anonymous nodes
   struct in6_addr          anonymous_address;
   // The id number of queue we are using, needs to be global for all to use it
          int               queueid;
   //Our IPv6 address
          char              our_ipv6_addr_name[INET6_ADDRSTRLEN+1];
   //The ipv4 address of the tunnel endpoint
          char              tunnelip[INET_ADDRSTRLEN+1];
   // Loop variable telling us to stop
          int               main_done         = PLZCONTINUE;

   // The total number of sources for our probes
   int    number_of_sources        = ZERO;
   // The total number of edges found
   int    number_of_edges          = ZERO;
   // The number in the global stop table
   int    number_of_global_stops   = ZERO;
   // The number of total nodes in the alias list
   int    number_of_nodes          = ZERO;
   // Used to store the pid of the probe generator
   int    probe_generator_pid      = ZERO;
   // Used to store the pid of the sys log facility
   int    sys_log_pid              = ZERO;
   // Used to store the pid of the probe receiver
   int    probe_receive_pid        = ZERO;
   // Tells us if we are main
   int    is_main                  = TRUE;
   // This records our checkpoint time
   time_t check_point_time         = ZERO;
   // The number we are using in the anonymous nodes
   unsigned int  anonymous_number  = ZERO;
   // The number of packets to be sent for each probe
   int    NUMBER_OF_PACKETS        = ZERO;
   // The minimum number of responses we will consider valid
   int    MIN_RESPONSE             = ZERO;
   // The maximum amount of anonymous nodes before we stop probing
   int    MAX_ANONYMOUS            = ZERO;
   // The default value using the cdf func to determine our starting hop count
   double DEFAULT_P_VALUE          = 0.0;
   // The time we will wait for a response
   int    WAIT_TIME                = ZERO;
   // How much time should elapse before we check point
   int    CHECK_POINT_TIME         = ZERO;

/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/

 void main_handler(int sig) {

    int pid     = ZERO;
    int status  = ZERO;
```

```
    if (is_main == TRUE) {

       printf("Probe system received the message and is stopping, " \
              "Please wait!\n");
       main_done = STOP;

    }//if (is_main == TRUE) {


}// void main_handler(int sig) {

 void child_handler(int sig) {

    char textbuffer[MAX_MESSAGE];  // Buffer to hold our message
    int  status;
    int  pid         = ZERO;


    pid = wait(&status);
    if (pid == sys_log_pid) {

       fprintf(stderr,"PROBE MAIN: Sys log stopped, if this is unplanned " \
                      "check log\n");
       fprintf(stderr,"PROBE MAIN: The sys log process stopped with " \
                      "status: %i\n", status);
       sys_log_pid = ZERO;
       if (TROUBLESHOOT != 99) {

          // If sys-log stopped we don't record so we should stop
          main_done   = STOP;

       }//if (TROUBLESHOOT != 99) {

    } else if (pid == probe_generator_pid) {

       fprintf(stderr,"PROBE MAIN: Probe Generator stopped, if this is " \
                      "unplanned check log\n");
       snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: The probe generator" \
                      " stopped with status: %i", status);
       call_sys_log(textbuffer);
       probe_generator_pid = ZERO;
       if (TROUBLESHOOT != 99) {

          main_done = STOP;       // If generator stopped no reason to continue

       }//if (TROUBLESHOOT != 99) {

    } else if (pid == probe_receive_pid) {

       fprintf(stderr,"PROBE MAIN: Probe Receiver stopped, if this is " \
                      "unplanned check log\n");
       snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: The probe receiver" \
                                         " stopped with status: %i", status);
       call_sys_log(textbuffer);
       probe_receive_pid = ZERO;
       if (TROUBLESHOOT != 99) {

          main_done = STOP;   // If receiver stopped no reason to continue

       }//if (TROUBLESHOOT != 99) {

    } else {

       snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: Received a signal " \
                "but pid did not match any running process, status = " \
                ": %i\n", status);
       call_sys_log(textbuffer);
```

```
    }//if (pid == sys_log_pid) {

}//void child_handler(int sig) {

/*****************************************************************************
 *
 * Main function
 *
 *****************************************************************************/

int main(int argc, char *argv[]){

   /*****************************************************************************
   *
   * Define variables needed by main
   *
   *****************************************************************************/

   // We may need this by the messaging system to check status
   struct                   msqid_ds msqid_ds, *buf;

                            buf           = &msqid_ds;
   // Flag to tell if any of the processes should continue or quit
   int                      continue_flag = ZERO;

   // Used by main to store destination addresses
   struct in6_addr          destination;
   // Flag to tell us if we had an error forking
   int                      fork_error    = FALSE;
   // PID returned by fork command
   int                      forked_pid    = ZERO;
   // Hop limit used by main
   int                      hop_limit     = ZERO;
   // This will be the value used to determine our queue id
   key_t                    key           = KEY;

   libnet_t                 *packet_handle;   //Initial handle for our packet
   // Used by main to store payload type, ICMP or UDP
   int                      payload_type  = ZERO;
   // The value of our receive op!!
   int                      rcvsuccess    = ZERO;
   // The value of our send operation!!
   int                      sendsuccess   = ZERO;
   // Used by main to store source addresses
   struct in6_addr          source;
   // Used by main to store the source port number
   int                      source_port   = ZERO;
   // Buffer to hold our messages for sys-log
   char                     textbuffer[MAX_MESSAGE];
   // General counter for loops
   int                      count         = ZERO;
   // Seed for our random number generator
   int                      seconds       = ZERO;


   /*****************************************************************************
   *
   * Set up initial vales like empty, and seed for the random number generator
   *
   *****************************************************************************/

   // Get value from system clock and place in seconds variable to use as seed.
   check_point_time = time(&seconds);
   //Convert seconds to a unsigned integer and seed.
   srand((unsigned int) seconds);

   int test = inet_pton(AF_INET6, "fe80::0", &empty);
   if (test <= ZERO){
```

```
   fprintf(stderr,"PROBE MAIN: Error unable to initialize " \
                  "values \"empty\"\n");
   return -1;

}//if (test <= ZERO){

/*************************************************************************
*
* Get our IPv6 address and the tunnel endpoint IPv4 address from startup
* If we did not get them on startup, error out and give usage details
*
*************************************************************************/

if (argc < 9) {

   printf("Probe Engine Program!\n");
   printf("Usage: %s [A] [B] [C] [D] [E] [F] [G] [H]\n",argv[0]);
   printf(" A = Tunnel end point ipv4 address\n");
   printf(" B = Your IPv6 address\n");
   printf(" C = Number of packets per probe\n");
   printf(" D = The minimum number of packets for valid response\n");
   printf(" E = The maximum number of anonymous nodes before we stop " \
          "probing out\n");
   printf(" F = The maximum number of seconds to wait for a response" \
          " before considering it anonymous\n");
   printf(" G = The default value using the cdf function to determine " \
          "our starting hop count\n");
   printf(" H = The number of minutes to run before check pointing\n");
   printf("Example: %s 172.16.0.1 2200::211:2:0:0:2568 3 2 2 2 0.05" \
          " 60\n", argv[0]);
   printf("Thanks for playing!!\n");
   return -1;

 }//if (argc < 9) {

 printf("NOTE: IF YOU ARE NOT ROOT THIS WILL FAIL!!!\n");

 if (TROUBLESHOOT==1) {

    fprintf(stderr, "I am about to do the strncpy\n");
    fprintf(stderr, "The value of argv[1] = %s\n", argv[1]);
    fprintf(stderr, "The value of argv[2] = %s\n", argv[2]);
    fprintf(stderr, "The value of argv[3] = %s\n", argv[3]);
    fprintf(stderr, "The value of argv[4] = %s\n", argv[4]);
    fprintf(stderr, "The value of argv[5] = %s\n", argv[5]);
    fprintf(stderr, "The value of argv[6] = %s\n", argv[6]);
    fprintf(stderr, "The value of argv[7] = %s\n", argv[7]);
    fprintf(stderr, "The value of argv[8] = %s\n", argv[8]);

 }//if (TROUBLESHOOT==1) {

 strncpy(tunnelip, argv[1], INET_ADDRSTRLEN);
 strncpy(our_ipv6_addr_name, argv[2], INET6_ADDRSTRLEN);
 char* endptr              = NULL;    // Used for number conversion
 NUMBER_OF_PACKETS         = (int)    strtoul(argv[3], &endptr, 10);
 MIN_RESPONSE              = (int)    strtoul(argv[4], &endptr, 10);
 MAX_ANONYMOUS             = (int)    strtoul(argv[5], &endptr, 10);
 WAIT_TIME                 = (int)    strtoul(argv[6], &endptr, 10);
 DEFAULT_P_VALUE           = (double) strtod( argv[7], &endptr);
 CHECK_POINT_TIME          = (int)    strtoul(argv[8], &endptr, 10);

 if (TROUBLESHOOT==1) {

   fprintf(stderr, "I am about to do the strncpy\n");
   fprintf(stderr, "The NUMBER_OF_PACKETS on the command line was :" \
                   " %i\n", NUMBER_OF_PACKETS);
   fprintf(stderr, "The MIN_RESPONSE on the command line was : %i\n", \
                   MIN_RESPONSE);
   fprintf(stderr, "The MAX_ANONYMOUS on the command line was : %i\n", \
```

```
                  MAX_ANONYMOUS);
   fprintf(stderr, "The WAIT_TIME on the command line was : %i\n", \
                  WAIT_TIME);
   fprintf(stderr, "The DEFAULT_P_VALUE on the command line was : %f\n", \
                  DEFAULT_P_VALUE);
   fprintf(stderr, "The CHECK_POINT_TIME on the command line was : %i\n", \
                  CHECK_POINT_TIME);

 }//if (TROUBLESHOOT==1) {

 if (NUMBER_OF_PACKETS <= ZERO) {

    printf("The number of packets is invalid\n");
    return -1;

 } else if ((MIN_RESPONSE > NUMBER_OF_PACKETS) || (MIN_RESPONSE <= ZERO)) {

    printf("The number of responses is invalid\n");
    return -1;

 } else if (MAX_ANONYMOUS <= ZERO) {

    printf("The number for maximum anonymous is invalid\n");
    return -1;

 } else if ((DEFAULT_P_VALUE < 0.0) || (DEFAULT_P_VALUE > 1.0) ) {

    printf("The number DEFAULT_P_VALUE is invalid\n");
    return -1;

 } else if (WAIT_TIME < ZERO) {

    printf("The number WAIT_TIME is invalid\n");
    return -1;

 } else if (CHECK_POINT_TIME <= ZERO) {

    printf("The number CHECK_POINT_TIME is invalid\n");
    return -1;

 }//if (NUMBER_OF_PACKETS <= ZERO) {

 // We need to convert it to seconds
 CHECK_POINT_TIME = CHECK_POINT_TIME * 60;

 /*************************************************************************
 *
 * Set up the message queue for all to use
 *
 *************************************************************************/

 queueid = msgget(key, 0766 | IPC_CREAT);    // Get our message queue
 if (queueid == IPC_ERROR) {

   fprintf(stderr,"PROBE MAIN: ERROR: Message Get Failed!!!!\n");
   print_IPC_error("PROBE MAIN");
   return -1;

} else {

   printf("PROBE MAIN:Message get succeeded, Queue =  %i\n", queueid );

}//if (queueid == IPC_ERROR) {

/*************************************************************************
*
* Kick off all the other functions
*
*************************************************************************/
```

```
is_main    = TRUE;
fork_error = FALSE;

/*****************************************************************************
*
* Kick off syslog
*
*****************************************************************************/
forked_pid = fork();

if (forked_pid == ZERO) {

   //We are the child
   is_main = FALSE;
   sys_log(); //We forked successfully we are the child!!!

} else if (forked_pid < ZERO) {

   fprintf(stderr,"PROBE MAIN: ERROR: trying to fork sys log, return = " \
                  "%i\n", forked_pid);
   fork_error = TRUE;

} else {

   //We are main and everything worked
   sys_log_pid = forked_pid;
   forked_pid  = ZERO;
   if (TROUBLESHOOT == 1){

      fprintf(stderr,"The pid for sys log is: %i\n",sys_log_pid);

   }//if (TROUBLESHOOT == 1){


}//if (forked_pid == ZERO) {


/*****************************************************************************
*
* Kick off probe receive
*
*****************************************************************************/

if (is_main && fork_error == FALSE) {


   forked_pid = fork();

   if (forked_pid == ZERO) {

      //We are the child
      is_main = FALSE;

      probe_receive(); //We forked successfully we are the child!!!

   } else if (forked_pid < ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: trying to " \
                  "fork probe receive, return = %i", forked_pid);
         call_sys_log(textbuffer);
         fork_error = TRUE;

   } else {

         //We are main and everything worked
         probe_receive_pid = forked_pid;
         if (TROUBLESHOOT == 1){
```

```
            fprintf(stderr,"The pid for probe receive is: %i\n", \
                           probe_receive_pid);

         }//if (TROUBLESHOOT == 1){
         forked_pid  = ZERO;

   }//if (forked_pid == ZERO) {

}//   if (is_main) {

/*****************************************************************************
*
* Kick off probe generator
*
*****************************************************************************/

if (is_main && fork_error == FALSE) {

   forked_pid = fork();

   if (forked_pid == ZERO) {

      //We are the child
      is_main = FALSE;
      probe_generator(); //We forked successfully we are the child!!!

   } else if (forked_pid < ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: trying to " \
                  "fork probe generator, return = %i", forked_pid);
         call_sys_log(textbuffer);
         fork_error = TRUE;

   } else {

         //We are main and everything worked
         probe_generator_pid = forked_pid;
         forked_pid  = ZERO;
         if (TROUBLESHOOT == 1){

            fprintf(stderr,"The pid for probe generator is: %i\n", \
                           probe_generator_pid);

         }//if (TROUBLESHOOT == 1){

   }//if (forked_pid == ZERO) {

}//   if (is_main) {

/*****************************************************************************
*
* Kick off main process
*
*****************************************************************************/

if (is_main && fork_error == FALSE) {

   // The success or failure of startup
   int    startup      = ZERO;
   // This causes us to loop until we are done
   int    done         = PLZCONTINUE;

   if ((sys_log_pid == ZERO) || (probe_generator_pid == ZERO) ||
      (probe_receive_pid == ZERO)){

      fprintf(stderr, "We had a bad startup, shutting down!!!\n");
      startup = 2;

   }//if ((sys_log_pid == ZERO) || (probe_generator_pid == ZERO) ||
```

```
if (startup == ZERO) {

   startup = load_sources();
   if (startup != ZERO) {

      fprintf(stderr, "Error loading sources!\n");

   }//if (startup != ZERO) {


}//if (startup == ZERO) {
if (startup == ZERO) {

   startup = load_edges();
   if (startup != ZERO) {

      fprintf(stderr, "Error loading edges!\n");

   }//if (startup != ZERO) {

}//if (startup == ZERO) {
if (startup == ZERO) {

   startup = load_stop_list();
   if (startup != ZERO) {

      fprintf(stderr, "Error loading stop list!\n");

   }//if (startup != ZERO) {

}//if (startup == ZERO) {
if (startup == ZERO) {

   startup = load_probe_list();
   if (startup != ZERO) {

      fprintf(stderr, "Error loading probe list!\n");

   }//if (startup != ZERO) {

}//if (startup == ZERO) {

/************************************************************************
*
* Kick off the signal handlers
*
************************************************************************/

signal(SIGCHLD, child_handler);
signal(SIGQUIT, main_handler);

/************************************************************************
*
* Setup before we begin probing
*
************************************************************************/

printf("Beginning setup of all the initial values before scanning!\n");

/************************************************************************
*
* Check to see if our sources have Zero h values, if so determine
* initial h values
*
************************************************************************/

if (TROUBLESHOOT==1) {
```

```
   fprintf(stderr,"Beginning the loop, number of sources = %i\n", \
                  number_of_sources);

}//if (TROUBLESHOOT==1) {

if (startup == ZERO) {

    for(count=ZERO;count<number_of_sources;count++) {

      printf("Calculating the initial h_value for source %i\n", count);

      if (source_list[count].h_value == ZERO) {

         if (TROUBLESHOOT==1) {

            fprintf(stderr,"Inside the loop processing source:" \
                           "\n", count);

         }//if (TROUBLESHOOT==1) {

         if (source_list[count].number_of_values == ZERO) {

            /*********************************************************
             *
             * If we are here we have nothing and we need to ping to get
             * our values
             *
             *********************************************************/

             if (TROUBLESHOOT==1) {

                fprintf(stderr,"source_list[count].number_of_values "\
                               "== ZERO\n");

             }//if (TROUBLESHOOT==1) {

             struct remaining_probes *temp_probe_pointer;
             temp_probe_pointer = source_list[count].probe_next;
             int run_number = ZERO;

             if (temp_probe_pointer != NULL) {

                while(temp_probe_pointer != NULL) {

                  int our_hop_count = 1;
                  int sequence      = our_hop_count;
                  int reached       = FALSE;

                  while(reached == FALSE) {

                     /*************************************************
                     *
                     * Prep the message to send
                     *
                     *************************************************/

                     run_number = rand() % (HIGH - LOW + 1) + LOW;
                     message_to_generator = (struct generator_buffer*)
                                            (sendbuffer.text);
                     message_to_generator->generator_cont  = PLZCONTINUE;
                     message_to_generator->source          =
                                             source_list[count].address;
                     message_to_generator->destination     =
                                            temp_probe_pointer->address;
                     message_to_generator->hop_limit       =
                                            our_hop_count +
                                           source_list[count].hop_count;
                     message_to_generator->protocol        =
                                           IPPROTO_ICMPV6 + SOURCEROUTE;
```

```
message_to_generator->port            = run_number;
message_to_generator->sequence        = sequence;

if (TROUBLESHOOT==1) {

   fprintf(stderr, "About to send the message to " \
                   "the generator:\n");
   fprintf(stderr, "The value of generator_cont" \
           " = %i\n", \
           message_to_generator->generator_cont);
   char  temp_addr_name[INET6_ADDRSTRLEN+1];
   inet_ntop(AF_INET6,
             &message_to_generator->source,
             temp_addr_name, INET6_ADDRSTRLEN);
   fprintf(stderr,"The source address = %s\n", \
                  temp_addr_name);
   inet_ntop(AF_INET6,
             &message_to_generator->destination,
             temp_addr_name, INET6_ADDRSTRLEN);
   fprintf(stderr, "The destination address = " \
           "%s\n", temp_addr_name);
   fprintf(stderr,"The hop limit = %i\n", \
           message_to_generator->hop_limit);
   fprintf(stderr,"The port number = %i\n", \
           message_to_generator->port);
   fprintf(stderr,"The sequence number = %i\n",\
           message_to_generator->sequence);

}//if (TROUBLESHOOT==1) {

/***********************************************
*
* Send the message
*
***********************************************/

sendbuffer.type = GENERATOR;

sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                     sizeof(struct generator_buffer),
                     IPC_NOWAIT);

if(sendsuccess != ZERO){

   fprintf(stderr,"PROBE MAIN: ERROR: Message send"\
                  " failed.  Error: ");
   print_IPC_error("PROBE MAIN");

} else {

   char  temp_addr_name[INET6_ADDRSTRLEN+1];
   inet_ntop(AF_INET6,
             &message_to_generator->destination,
             temp_addr_name, INET6_ADDRSTRLEN);
   snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: "\
            "Message to probe generator for dest " \
            "%s for src %i with h_limit %i", \
            temp_addr_name, count, \
            message_to_generator->hop_limit);
   call_sys_log(textbuffer);

}//if(sendsuccess != ZERO){

/***********************************************
*
* We wait for the response, if no response in wait
* time we move on
*
***********************************************/
```

```
time_t sendtime = time (NULL);
int response       = FALSE;
int response_count = ZERO;
int NotEnoughHops  = FALSE;

while (((time(NULL) - sendtime) < WAIT_TIME) &&
       (response == FALSE)){

   //Get the message off the queue, do not block
   rcvsuccess = msgrcv(queueid, (void *)
                    &recvbuffer,
                    sizeof(struct receive_buffer),
                    RECEIVE, IPC_NOWAIT);

   if (TROUBLESHOOT==1) {

      fprintf(stderr, "rcvsuccess = %i\n", \
                    rcvsuccess);
      if (rcvsuccess == IPC_ERROR) {

         fprintf(stderr, "errno = %i\n", errno);

      }//if (recvsuccess == IPC_ERROR) {

   }//if (TROUBLESHOOT==1) {

   if(rcvsuccess == IPC_ERROR  && errno != ENOMSG) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN:"\
              " ERROR: Message Receive failed.");
      call_sys_log(textbuffer);
      print_IPC_error("PROBE MAIN");

   }  else if (rcvsuccess != IPC_ERROR) {

      message_from_receiver =
         (struct receive_buffer*) (recvbuffer.text);

      if (TROUBLESHOOT==1) {

         fprintf(stderr, "The message received " \
                         "is:\n");
         char  temp_addr_name[INET6_ADDRSTRLEN+1];
         inet_ntop(AF_INET6,
                   &message_from_receiver->source,
                   temp_addr_name,
                   INET6_ADDRSTRLEN);
         fprintf(stderr,"The received source " \
                  "address = %s\n", temp_addr_name);
         inet_ntop(AF_INET6,
                &message_from_receiver->destination,
                temp_addr_name, INET6_ADDRSTRLEN);
         fprintf(stderr, "The received destination"\
                  " address = %s\n", temp_addr_name);
         fprintf(stderr,"The received protocol" \
                  " = %i\n",
                  message_from_receiver->protocol);
         inet_ntop(AF_INET6,
            &message_from_receiver->original_source,
            temp_addr_name, INET6_ADDRSTRLEN);
         fprintf(stderr,"The received original"\
                  " source address = %s\n", \
                  temp_addr_name);
         inet_ntop(AF_INET6,
       &message_from_receiver->original_destination,
       temp_addr_name, INET6_ADDRSTRLEN);
         fprintf(stderr,"The received original "\
                  "destination address = %s\n", \
```

```
                temp_addr_name);
        if(message_from_receiver->original_protocol
            == IPPROTO_ICMPV6) {

           fprintf(stderr,"The received original" \
                   " protocol was ICMPv6\n");

}else if(message_from_receiver->original_protocol
            == IPPROTO_UDP) {

           fprintf(stderr,"The received original" \
                   " protocol was UDP\n");

        } else {

           fprintf(stderr,"The received original" \
                   " protocol did not match it =" \
                   " %i\n",
         message_from_receiver->original_protocol);

        }//if (message_from_receiver->original_prot
        fprintf(stderr,"The received original hop"\
                " limit = %i\n",
        message_from_receiver->original_hop_limit);
        if (message_from_receiver->icmp_type ==
            TIME_EXCEEDED) {

           fprintf(stderr,"The received icmp type"\
                   " = TIME Exceeded\n");

        } else if(message_from_receiver->icmp_type
                     == DESTINATION_UNREACHABLE) {

           fprintf(stderr,"The received icmp type"\
                   " = Destination unreachable\n");

        } else {

           fprintf(stderr,"The received icmp type"\
                   " was unknown it = %i\n", \
                 message_from_receiver->icmp_type);

        }//if (message_from_receiver->icmp_type ==
        fprintf(stderr,"The received icmp code ="\
                " %i\n", \
                message_from_receiver->icmp_code);
        fprintf(stderr,"The received icmp ident ="\
                " %i\n", \
                message_from_receiver->icmp_ident);
        fprintf(stderr,"The icmp ident we want is"\
                " = %i\n", run_number);
        fprintf(stderr,"The received icmp "\
                "sequence = %i\n", \
             message_from_receiver->icmp_sequence);
        fprintf(stderr,"The icmp sequence we want"\
                       " is = %i\n", sequence);
        fprintf(stderr,"The received original " \
                "source port = %i\n", \
      message_from_receiver->original_source_port);
        fprintf(stderr,"The received original " \
                "dest port = %i\n", \
        message_from_receiver->original_dest_port);
        fprintf(stderr,"The received original " \
                "route seg left = %i\n", \
 message_from_receiver->original_route_seg_left);
        inet_ntop(AF_INET6,
       &message_from_receiver->original_route_addr,
                 temp_addr_name, INET6_ADDRSTRLEN);
        fprintf(stderr,"The received original " \
```

```
                "route address = %s\n", \
                temp_addr_name);
            fprintf(stderr,"The received original "\
                "icmp type = %i\n", \
            message_from_receiver->original_icmp_type);
            fprintf(stderr,"The received original " \
                "icmp code = %i\n", \
            message_from_receiver->original_icmp_code);

        }//if (TROUBLESHOOT==1) {

        if((message_from_receiver->icmp_type ==
            DESTINATION_UNREACHABLE) &&
          (message_from_receiver->icmp_code !=
            PORT_UNREACHABLE)) {

            our_hop_count = MAX_HOPS;
            response = TRUE;
            char  temp_addr_name[INET6_ADDRSTRLEN+1];
            inet_ntop(AF_INET6,
                    &message_to_generator->destination,
                    temp_addr_name, INET6_ADDRSTRLEN);
            snprintf(textbuffer, MAX_MESSAGE, "PROBE"\
                     " MAIN: Received unreachable "\
                      "message for dest %s for source"\
                      " %i", temp_addr_name, count);
            call_sys_log(textbuffer);

        } else if (message_from_receiver->protocol ==
                    IPPROTO_ICMPV6 &&
            message_from_receiver->icmp_type ==
                    TIME_EXCEEDED &&
            run_number ==
                  message_from_receiver->icmp_ident &&
            message_from_receiver->icmp_sequence ==
                  sequence) {

            char  temp_addr_name[INET6_ADDRSTRLEN+1];
            inet_ntop(AF_INET6,
                  &message_from_receiver->source,
                  temp_addr_name, INET6_ADDRSTRLEN);
            snprintf(textbuffer, MAX_MESSAGE, "PROBE"\
                     " MAIN: Received time exceeded " \
                     "msg from dest %s for src %i",\
                     temp_addr_name, count);
            call_sys_log(textbuffer);

            if (TROUBLESHOOT==1) {

               fprintf(stderr,"We matched, time "\
                       "exceeded on our message\n");

            }//if (TROUBLESHOOT==1) {

            /***************************************
            *
            * Check to make sure we have enough hops
            * in our estimation of the distance of the
            * source router
            * If not bump up the hop count to it
            *
            ***************************************/

        if (message_from_receiver->original_route_seg_left
             > ZERO) {

                NotEnoughHops = TRUE;
                snprintf(textbuffer,MAX_MESSAGE,"PROBE"\
                         " MAIN: Source %i did not "\
```

```
                "have enough hops bumping"\
                " by one", count);
      call_sys_log(textbuffer);

      if (TROUBLESHOOT == 20) {

         fprintf(stderr, "This source did "\
                 "not have enough hops, "\
                 "bumping it up by one!\n");

      }//if (TROUBLESHOOT == 20) {

   }//if (message_from_receiver->original_rout

   response_count++;

   /****************************************
   *
   * Don't want to exit the loop until we
   * process all the packets we sent or
   * time out
   *
   ****************************************/

   if (response_count >= NUMBER_OF_PACKETS) {

      response = TRUE;

   }//if (response_count >= NUMBER_OF_PACKETS)

} else if (message_from_receiver->protocol ==
           IPPROTO_ICMPV6 &&
  (cmpaddr(&message_from_receiver->source,
           &temp_probe_pointer->address,
           128) == ZERO) &&
   run_number ==
          message_from_receiver->icmp_ident &&
   message_from_receiver->icmp_sequence ==
          sequence) {

   char  temp_addr_name[INET6_ADDRSTRLEN+1];
   inet_ntop(AF_INET6,
             &message_from_receiver->source,
             temp_addr_name,
             INET6_ADDRSTRLEN);
   snprintf(textbuffer, MAX_MESSAGE, "PROBE"\
            " MAIN: Received response msg " \
            "from dest %s for src %i", \
            temp_addr_name, count);
   call_sys_log(textbuffer);

   response_count++;

   /****************************************
   *
   * If we get the minimum number for a valid
   * response then we want to reocrd it
   *
   ****************************************

   if (response_count >= MIN_RESPONSE) {

      reached = TRUE;

   }// if (response_count >= MIN_RESPONSE) {

   /****************************************
   *
   * Don't want to exit the loop until we
```

```
                    * process all the packets we sent or
                    * time out
                    *
                    ****************************************/

                    if (response_count >= NUMBER_OF_PACKETS) {

                       response = TRUE;

                    }//if (response_count >= NUMBER_OF_PACKETS)

                  } else {

                    char  temp_addr_name[INET6_ADDRSTRLEN+1];
                    inet_ntop(AF_INET6,
                              &message_from_receiver->source,
                              temp_addr_name,
                              INET6_ADDRSTRLEN);
                    snprintf(textbuffer, MAX_MESSAGE,"PROBE"\
                             " MAIN: Received unknown packet" \
                             " from %s destination", \
                             temp_addr_name);
                    call_sys_log(textbuffer);

                  }//if(message_from_receiver->protocol == IPPROT

                }//if(rcvsuccess == IPC_ERROR  && errno != ENOMSG

             }//while ((curtime - time(NULL) < WAIT_TIME) && (res

             if (our_hop_count < (MAX_HOPS/4) && reached ==
                 FALSE &&
                 NotEnoughHops == FALSE) {

                our_hop_count++;

             } else if (NotEnoughHops == TRUE) {

                source_list[count].hop_count++;

             } else {

                reached = TRUE;

             }//if (our_hop_count < MAX_HOPS) {

          }//while(reached == FALSE) {

          if ((our_hop_count < (MAX_HOPS/4)) &&
              (our_hop_count >= ZERO)) {

             source_list[count].hop_length[our_hop_count]++;
             source_list[count].number_of_values++;

          }//if (our_hop_count < MAX_HOPS) {
          temp_probe_pointer = temp_probe_pointer->next;

       }//while(temp_probe_pointer != NULL) {

    } else {

       snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: " \
                "Unable to determine initial h-value " \
                "for source %i list is empty", count);
       call_sys_log(textbuffer);

    }//if (temp_probe_pointer != NULL) {

    /*********************************************************
```

```
            *
            * We have pinged all the destinations now calcualte the
            * h_value
            *
            ************************************************************/

            if (source_list[count].number_of_values > ZERO) {

               source_list[count].h_value =
                    calc_p_value(source_list[count].hop_length,
                                 source_list[count].number_of_values);

            }//if (source_list[count].number_of_values > ZERO) {

         } else {

            /************************************************************
            *
            * If we are here we have an initial list and just need to
            * calculate
            *
            ************************************************************/

            if (source_list[count].number_of_values > ZERO) {

               source_list[count].h_value =
                    calc_p_value(source_list[count].hop_length,
                                 source_list[count].number_of_values);

            }//if (source_list[count].number_of_values > ZERO) {

         }//if (source_list[count].number_of_values == ZERO) {

      }//if (source_list[count].h_value == ZERO) {

   }//for(count=ZERO;count<number_of_sources;count++) {

}//if (startup == ZERO) {

/*************************************************************************
*
* This is the main routine that accomplishes all the tasks
*
*************************************************************************/

if (TROUBLESHOOT==1) {

   fprintf(stderr,"We are about to setup for the main loop\n");

}//if (TROUBLESHOOT==1) {

if (startup == ZERO) {

   /*********************************************************************
   *
   * Build processing list and initialize it to empty
   *
   *********************************************************************/

   if (TROUBLESHOOT==1) {

      fprintf(stderr,"We are about to empty the processing list\n");

   }//if (TROUBLESHOOT==1) {

   struct processing_source processing_list[number_of_sources];
   struct processing_source *processing_list_pointer = processing_list;
   for(count=ZERO;count<number_of_sources;count++) {
```

```
   processing_list[count].destination             = empty;
   processing_list[count].list_number             = ZERO;
   processing_list[count].direction               = ZERO;
   processing_list[count].id                      = ZERO;
   processing_list[count].sequence                = ZERO;
   processing_list[count].hop_count               = ZERO;
   processing_list[count].time_sent               = ZERO;
   processing_list[count].response_number         = ZERO;
   processing_list[count].previous_node           = empty;
   processing_list[count].anonymous_response_count = ZERO;
   processing_list[count].generate                = FALSE;
   processing_list[count].bad                     = FALSE;

}//for(count=ZERO;count<number_of_sources;count++) {

if (TROUBLESHOOT==1) {

   fprintf(stderr,"We are about to fill the processing list\n");

}//if (TROUBLESHOOT==1) {

fill_list(processing_list);       //Fills the process list
printf("Done setting up begining the scans!\n");
if (TROUBLESHOOT==1) {

   fprintf(stderr,"We are about to generate the probes\n");

}//if (TROUBLESHOOT==1) {
generate_probes(processing_list); // Generates probes
printf("Please enter CTRL \\ to stop!!\n");
if (TROUBLESHOOT==1) {

   fprintf(stderr,"Entering the main loop\n");

}//if (TROUBLESHOOT==1) {

while (main_done == PLZCONTINUE){

   if (main_done == PLZCONTINUE) {

      if (check_pointer() == STOP) {

         main_done = STOP;

      }//if (check_pointer() == STOP) {

   }//if (main_done == PLZCONTINUE) {
   // Slows down our processing to prevent floods
   sleep(1);
   if (main_done == PLZCONTINUE) {

      if(process_receive(processing_list) == STOP) {

         main_done = STOP;

      }//if(process_receive(processing_list) == STOP) {

   }//if (main_done == PLZCONTINUE) {
   if (main_done == PLZCONTINUE) {

      fill_list(processing_list);      // Fill the process list
      generate_probes(processing_list); // Generates new probes
      if (main_done == PLZCONTINUE) {

         if(done_check(processing_list) == STOP){

            main_done = STOP;

         }//if(done_check(processing_list) == STOP){
```

```
        }//if (main_done == PLZCONTINUE) {

      }//if (main_done = PLZCONTINUE) {

   }//while (main_done == PLZCONTINUE){


} else {

   fprintf(stderr,"PROBE MAIN: ERROR in loading startup files. "\
                  " Shutting down!\n");

}//if (startup == ZERO) {

/*************************************************************************
*
* Check point if we were successfull in loading our files!!!
*
*************************************************************************/

if (startup == ZERO) {

   int temp = ckpt();
   if (temp != ZERO) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: Check point error");
      call_sys_log(textbuffer);
      fprintf(stderr, "PROBE MAIN: We had a failure check pointing, "\
                      "check the log for error\n");

   } else {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: Check point " \
                                       "successful");
      call_sys_log(textbuffer);
      fprintf(stderr,"%s\n", textbuffer);

   }//if (temp != ZERO) {

}//if (startup == ZERO) {

/*************************************************************************
*
* Kill the probe generator if still running
*
*************************************************************************/

if (probe_generator_pid != ZERO) {

   message_to_generator = (struct generator_buffer*) (sendbuffer.text);
   message_to_generator->generator_cont  = STOP;


  /*************************************************************************
  *
  * Send the message
  *
  *************************************************************************/

  sendbuffer.type = GENERATOR;
  sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                       sizeof(struct generator_buffer), IPC_NOWAIT);

  if(sendsuccess != ZERO){

      snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: Message " \
                                       "send failed to stop generator."\
                                       "  Error: ");
```

```
        call_sys_log(textbuffer);
        print_IPC_error("PROBE MAIN");

    } else {

        snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: Message was sent to"\
                                         " terminate generator");
        call_sys_log(textbuffer);

    }//if(sendsuccess != ZERO){

}// if (probe_generator_pid != ZERO) {

/*************************************************************************
*
* Wait for all the probes to return before killing the receiver
* NOTE: Loop used instead of sleep(10) because signals wake up sleep
*
*************************************************************************/

int count = 0;
if (probe_receive_pid != ZERO) {

   fprintf(stderr,"PROBE MAIN: Waiting for 10 seconds for all probes to"\
                  " return before killing probe receive\n");
   while(count <= 10){

      count++;
      sleep(1);

   }//while(count <= 5){

}//if (probe_receive_pid != ZERO) {

if (probe_receive_pid != ZERO) {

 int temp = kill(probe_receive_pid, SIGKILL);
 if (temp < ZERO){

    snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: Message send "\
                                     "failed to stop receiver.  Error: ");
    call_sys_log(textbuffer);
    print_IPC_error("PROBE MAIN");

 }//if (temp < ZERO){

}//if (probe_receive_pid != ZERO) {

/*************************************************************************
*
* Wait for the probe receive to stop
*
*************************************************************************/

count = 0;
while((probe_receive_pid  != ZERO) && (count <= 5)){

   fprintf(stderr,"PROBE MAIN: Waiting for probe receive to finish\n");
   count++;
   sleep(1);

}//while((probe_receive_pid  != ZERO) && (count <= 5)){

/*************************************************************************
*
* Wait for the probe generator to stop
*
*************************************************************************/
```

```
      count = ZERO;
      while((probe_generator_pid  != ZERO) && (count <= 5)){

         snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: Waiting for probe " \
                                          "generator to finish");
         printf("%s\n",textbuffer);
         count++;
         sleep(1);

      }//while((probe_generator_pid  != ZERO) && (count <= 5)){

      /*************************************************************************
      *
      * Kill the sys log
      *
      *************************************************************************/
      sendbuffer.type = SYSLOG;
      snprintf(sendbuffer.text, 5, "QUIT");

      sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                           sizeof(sendbuffer.text), IPC_NOWAIT);
      if(sendsuccess != ZERO){

         fprintf(stderr,"PROBE MAIN: ERROR: Unable to tell SYS LOG to STOP\n");
         print_IPC_error("PROBE MAIN");

      }//if(sendsuccess != ZERO){

      /*************************************************************************
      *
      * Wait for the sys log to stop
      *
      *************************************************************************/

      count = 0;
      while((sys_log_pid  != ZERO) && (count <= 5)){

         printf("PROBE MAIN: Waiting for sys log to finish\n");
         count++;
         sleep(1);

      }//while((sys_log_pid  != ZERO) && (count <= 5)){

      //Destroy the queue and clean up
      int destroy = msgctl(queueid, IPC_RMID, buf);
      if(destroy != ZERO){

         fprintf(stderr,"PROBE MAIN: ERROR: Unable to delete the queue, " \
                        "error number = %i\n", destroy);
         print_IPC_error("PROBE MAIN");

      } else {

         fprintf(stderr,"PROBE MAIN: Message QUEUE destroyed\n");

      }//if(destroy != ZERO){

   }// if (is_main && fork_error == FALSE) {

}//int main(int argc, char *argv[])
```

PROBE_MAIN.H

```
/*****************************************************************************
 * This is the header file of the main program of the probing engine
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_main.h
 *
 *****************************************************************************/
#ifndef INCLUSION_GUARD_PROGRAM_PROBE_MAIN
#define INCLUSION_GUARD_PROGRAM_PROBE_MAIN

#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <libnet.h>
#include <pcap.h>

#define GENERATOR               2876   // This is the Message type used to talk
                                       //  to the packet generator
#define SYSLOG                  3567   // This is the message type of syslogger
#define RECEIVE                 4328   // This is message type from receiver
#define MAIN                    5638   // This is the message type to main
#define INTERFACE               1478   // This is the message type of interface
#define ZERO                    0      // The constant zero
#define MAXQUEUE                48     // The size of messages for the queue
#define KEY                     3245   // Key used to determine our queue id
#define PLZCONTINUE             1      // Tells other functions to stop or go
#define STOP                    0      // Tells other functions to stop or go
#define IPC_WAIT                0      // This tells the IPC system to block
                                       // until there is a message
#define MAX_MESSAGE             100    // The max size of message to sys logger
#define MAX_SIZE                200    // The max size of IPC message buffer
#define SOURCEROUTE             100
#define OUTDEVICE               "eth0" //   Device we will send packets out on
#define TRUE                    1      // Value used for true in the routine
#define FALSE                   0      // Value used for false in the routine
#define BADMASK                 2222   // If we pass a bad mask to cmp addr
#define TUNNEL_PROTO            41     // Type of packet for IPv6 tunnel
#define DESTINATION_UNREACHABLE 1      // The value of a dest unreachable msg
#define ADMISTRATIVE            1      // The value of admin unreachable msg
#define ROUTE_UNREACHABLE       0      // The value of route unreachable msg
#define PORT_UNREACHABLE        4      // The code for an unreachable message
#define TIME_EXCEEDED           3      // The value of a time exceeded message
#define ECHO_REQUEST            128    // The value of an echo reply
#define ECHO_REPLY              129    // The value of an echo reply
#define MAX_LINE_LENGTH         80     // The maximum length of file we read in
#define MAX_EDGES               150000 // The max number of edges we can store
#define FOUND                   0      // We found the value during a search
#define NOT_FOUND               1      // We did not find it in the list
#define NOT_FOUND_AND_INSERTED  1      // We did not find it but we inserted it
#define MAX_HOPS                64     // The max number of hops we will deal with
                                       // for p calc
#define LOW                     1      // The lowest random number we want
#define PORT_LOW                1025   // The lowest port to use for probes
#define HIGH                    32767  // The highest value for random gen
#define DATELENGTH              10     // Length we want to make  date string
#define NO_ORIGINAL_FILE        2      // This is the code if system wants to
                                       // save the old file to a backup
                                       // and the original is not there

   struct generator_buffer {  // Used to carry messages to the generator func
```

```
   // This tells the generator if it should shut down
   int                     generator_cont;
   // This is the address we want to use as the source of the probe
   struct in6_addr         source;
   // The is the address we want to probe
   struct in6_addr         destination;
   // The hop limit we want on the packet
   int                     hop_limit;
   // The protocol we want to use for this packet
   int                     protocol;
   // port number to use for requests also the id number in an icmp request
   int                     port;
   //The sequence number in an icmp request or the dest port of an UDP probe
   int                     sequence;


};


struct receive_buffer {  // Used to carry messages from the receiver

   struct in6_addr          source;    // Source address of packet
   struct in6_addr          destination; // Destination address of packet
   unsigned int             protocol;    // The protocol carried by packet
   struct in6_addr          original_source; // Original src addr of packet
   struct in6_addr          original_destination; // Orig dest addr of pkt
   unsigned int             original_protocol;  // Original proto of packet
   unsigned int             original_hop_limit;// remaining hop limit of pkt
   unsigned int             icmp_type;      // The icmp type of this message
   unsigned int             icmp_code;     // The icmp code of this message
   unsigned int             icmp_ident;    // The id num of icmp echo reply
   unsigned int             icmp_sequence; //Sequence num of icmp echo reply
   // The original source port of the sent packet
   unsigned int             original_source_port;
   // The original dest port of packet
   unsigned int             original_dest_port;
   // The # of segements left in original packet
   unsigned int             original_route_seg_left;
   // The next sgement address in route header
   struct in6_addr          original_route_addr;
   unsigned int             original_icmp_type; // The icmp type of message
   unsigned int             original_icmp_code; // The icmp code of message


};

struct messagebuffer {

   long                type;           //Identifies who is the receiver
   unsigned char       text[MAX_SIZE]; //The message we are sending

};

struct processing_source {

   struct in6_addr destination;              // destination we are probing
   struct in6_addr response_addr;            // Addr that responded to probe
   // Number from probe list this destination represents for later deletion
   int             list_number;
   int             direction;                // Direction we are probing
   int             id;                       // id number of latest probe
   int             sequence;              // sequence number of latest probe
   int             hop_count; // hop count for latest probe
   unsigned int    time_sent;                // time the last probe was sent
   int             response_number;          // Number of responses received
   struct in6_addr previous_node;            // previous node's address
   // Number of anonymous responses for latest probe
   int             anonymous_response_count;
```

```
   // flag telling the probe enerator to generate a new probe
   int             generate;
   // Flag telling us we started at the source, no need to turn around
   int             source_start;
   // Flag telling us we received a bad response and need to handle it
   int             bad;

} processing_source;


struct sources {                      //Maintains all the info for a source
  int hop_count;                   //This sources hop count from the tunnel
  struct in6_addr address;         //This sources address
  struct stops* stop_next;         //The local stop list for this source
  int    remaining_probe_counter;  //The number of remaining probes left
  //The list of remaining locations to probe
  struct remaining_probes* probe_next;
  int    h_value;                     //The initial hop value of our probes
  int    number_of_values;       //The number of hop lengths in the array
  //hop lengths we have recived back from our pings
  int    hop_length[MAX_HOPS];
} sources ;

struct edges {               //Maintains all the information for an edge
  struct in6_addr edge_v1;              //V1 of the edge
  struct in6_addr edge_v2;              //V2 of the edge
  struct in6_addr starting_source;      //The source for locating this edge
  //The final destination for probes when we located this edge
  struct in6_addr final_dest;
  int    hop_count;               //The hop count this edge was located at
} edges ;

struct global_stops { //This is the global stop set everyone uses
  struct in6_addr address;              //The address located
  struct in6_addr dest_address;   //The eventuial destination of this probe
} global_stops ;

struct alias_nodes {
  struct in6_addr original_address;
  struct in6_addr resolved_address;
} alias_nodes;

                       //NOTE: Linked list used here because list is small
struct stops {            //This is the local stop set for each source
  struct in6_addr address;              //The address we located
  struct stops* next;                   //Link to the next stop
} stops ;

struct remaining_probes {     //Addresses we need to probe with this engine
  struct in6_addr address;              //The address to probe
  struct remaining_probes* next;        //the next address on the list
} remaining_probes ;



 extern const  int    IPC_ERROR;    // The msg function calls -1 an error
 extern const  int    LIBNET_ERROR; // Libnet calls -1 an error
 // The id num of queue we are using, must be global since programs use it
 extern        int    queueid;

 extern        int    main_done;    // Loop variable telling us to stop
 extern        pcap_t *pcap_handle; // The handle to our pcap session

 //Our IPv6 address
 extern        char  our_ipv6_addr_name[INET6_ADDRSTRLEN+1];
 //The ipv4 address of the tunnel endpoint
 extern        char  tunnelip[INET_ADDRSTRLEN+1];
```

```
    // The total number of sources for our probes
    extern        int     number_of_sources;
    // The total number of edges from our probes
    extern        int     number_of_edges;
    // The total number of global dups
    extern        int     number_of_global_stops;
    // The number of total nodes in the alias list
    extern        int     number_of_nodes;
    // Pointer to the list of sources
    extern struct sources        *source_list;
    // Pointer to the list of edges
    extern struct edges          *edge_list;
    // Pointer to the alias list for alias resolution
    extern struct alias_nodes    *alias_list;
    // Pointer to the global stop set
    extern struct global_stops   *global_stop_list;
    // Pointer to the local stop set
    extern        void*          *probe_source_stop_list;
    // A flag address stating there is no address
    extern struct in6_addr       empty;
    // This records our checkpoint time
    extern        time_t         check_point_time;
    // Flag stating we are going in the positive direction
    extern const  int            positive;
    // Flag stating we are going in the negative direction
    extern const  int            negative;
    // address used to mark anonymous nodes
    extern struct in6_addr       anonymous_address;
    // The number we are using in the anonymous nodes
    extern unsigned int          anonymous_number;
    // The number of packets to send per probe
    extern        int            NUMBER_OF_PACKETS;
    // The minimum number of responses we will consider valid
    extern        int            MIN_RESPONSE;
    // The maximum amount of anonymous nodes before we stop probing
    extern        int            MAX_ANONYMOUS;
    // The default value using the cdf function to determine hop count
    extern        double         DEFAULT_P_VALUE;
    // the time we will wait for a response
    extern        int            WAIT_TIME;
    // How much time should elapse before we check point
    extern        int            CHECK_POINT_TIME;
    // Used to hold the current date
    extern        char           date[DATELENGTH];
    // Used to hold the number of edges deleted
    extern        int            deleted_edges;

#endif //INCLUSION_GUARD_PROGRAM_PROBE_MAIN
```

PROBE_MAIN_HELPER.C

```
/*****************************************************************************
 * This is the module contains the helper routines for main to run
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_main_helper.c
 *
 *****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <netinet/in.h>
#include "probe_main.h"
#include "probe_main_helper.h"
#include "probe_utils.h"
#include "sys_log.h"

#define TROUBLESHOOT 0           // 1 = fill_list      2 = generate_probes
                                 // 3 = process_packet
                                 // 4 = Process Source 5 = pull_from_list
                                 // ANY > 0 = sequential id
                                //           numbers for packets
#define REVERSE     TRUE
#define NO_REVERSE  FALSE

/*****************************************************************************
 *
 * Define externals and globals
 *
 *****************************************************************************/

   // Set by calls to the library functions to define errors encountered
   extern int errno;

   // Buffer to hold our message for syslog
          char              textbuffer[MAX_MESSAGE];

          int               probe_number_counter = ZERO;

/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/

  void process_packet(struct processing_source processing_list[]);
  int  process_source(struct processing_source processing_list[]);
  int  insert_on_list(const struct in6_addr address,
                      const int finding_source);
  void pull_from_list(int probe_source, int value_to_pull,
                      struct in6_addr our_address);
  void clear_from_processing_list(struct processing_source processing_list[],
                                  int probe_source);
  void move(struct processing_source processing_list[], int probe_source,
            int reverse);
  void change_direction(struct processing_source processing_list[],
                        int probe_source);

   int process_receive(struct processing_source processing_list[]) {

      process_packet(processing_list);
      return process_source(processing_list);
```

```
}//void process_receive(struct processing_source *processing_list[]) {


void fill_list(struct processing_source processing_list[]) {

   int count = ZERO;

   if (TROUBLESHOOT==1) {

      fprintf(stderr,"We are about to begin filling the list\n");

   }//if (TROUBLESHOOT==1) {

   /*************************************************************************
   *
   * Go through all the sources to see if they are ready for another
   * destination
   *
   *************************************************************************/
   for(count=ZERO; count < number_of_sources; count++) {

      if (TROUBLESHOOT==1) {

         fprintf(stderr,"Working on # %i on the list\n", count);
         fprintf(stderr,"The value of processing_list[count].list_number "\
                        "is: %i\n", processing_list[count].list_number);

      }//if (TROUBLESHOOT==1) {

      if(cmpaddr(&empty, &(processing_list[count].destination), 128)
         == ZERO) {

         /*********************************************************************
         *
         * If we are here then this source needs a new destination to probe
         * Only try to fill the list if there is a destination on the list
         *
         *********************************************************************/

         if (source_list[count].remaining_probe_counter > ZERO) {

            /*****************************************************************
            *
            * If we are here there was a destination on the list
            *
            *****************************************************************/

            struct remaining_probes *temp_probe_pointer;
            int    number_on_list =
                       source_list[count].remaining_probe_counter;
            int    one_we_want        =
                       ((int) (number_on_list - 1) *
                       (rand() / (RAND_MAX + 1.0)));
                   temp_probe_pointer = source_list[count].probe_next;
            int    temp_counter       = ZERO;

            if ((one_we_want < ZERO) || (one_we_want >= number_on_list)) {

               snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: Fill"\
                        " list selected a bad destination selected %i "\
                        " and max = %i, reset to 0",\
                        one_we_want, number_on_list);
               call_sys_log(textbuffer);
               one_we_want = ZERO;

            }//if ((one_we_want < ZERO) || (one_we_want >= number_on_list))

            if (TROUBLESHOOT==1) {
```

```
      fprintf(stderr,"The one we want on the list is %i for "\
                     "source %i\n",one_we_want, count);

   }//if (TROUBLESHOOT==1) {

   /************************************************************
   *
   * Grab a random one off the list so each source is probing a
   * different point
   *
   ************************************************************/

   while ((temp_counter < one_we_want) &&
          (temp_probe_pointer != NULL)) {

      temp_probe_pointer = temp_probe_pointer->next;
      temp_counter= temp_counter + 1;

   }//while (temp_counter < one_we_want && temp_probe_pointer != NU

   processing_list[count].destination =
                                  temp_probe_pointer->address;
   processing_list[count].response_addr = empty;
   processing_list[count].list_number   = one_we_want;
   processing_list[count].direction     = positive;
   processing_list[count].hop_count = source_list[count].h_value;
   processing_list[count].generate      = TRUE;
   processing_list[count].bad           = FALSE;
   if (processing_list[count].hop_count <= ZERO) {

      processing_list[count].source_start  = TRUE;
      processing_list[count].previous_node =
                                    source_list[count].address;

   } else {

      processing_list[count].source_start  = FALSE;
      processing_list[count].previous_node = empty;

   }//if (processing_list[count].hop_count <= ZERO) {
   if (TROUBLESHOOT==1) {

      fprintf(stderr,"Loaded processing list for source %i "\
                     "with the following:\n", count);
      char  temp_addr_name[INET6_ADDRSTRLEN+1];
      inet_ntop(AF_INET6, &(processing_list[count].destination),
                temp_addr_name, INET6_ADDRSTRLEN);
      fprintf(stderr,"Destination address: %s\n", temp_addr_name);
      fprintf(stderr,"List number %i\n", \
                      processing_list[count].list_number);
      fprintf(stderr,"Direction %i\n", \
                      processing_list[count].direction);
      fprintf(stderr,"Hop Count %i\n", \
                      processing_list[count].hop_count);
      fprintf(stderr,"Generate flag %i\n", \
                      processing_list[count].generate);

   }//if (TROUBLESHOOT==1) {

} else {

   if (TROUBLESHOOT==1) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: Fill list"\
                           " has nothing for source %i",count);
      call_sys_log(textbuffer);
      fprintf(stderr,"%s\n", textbuffer);

   }//if (TROUBLESHOOT==1) {
```

```
         }//if (source_list[probe_source].number_of_remaining_probes > ZERO)

      } else {

         if (TROUBLESHOOT==1) {

            fprintf(stderr,"Source # %i did not need anything\n", count);

         }//if (TROUBLESHOOT==1) {

      }//if(cmpaddr(&empty, &(processing_list[count].destination), 128)==ZER

   }//for(count=ZERO; count < number_of_sources; count++) {

   if (TROUBLESHOOT==1) {

      fprintf(stderr,"We are done filling the list\n");

   }//if (TROUBLESHOOT==1) {


}//void fill_list() {

void generate_probes(struct processing_source processing_list[]) {

   int count = ZERO;

   /*********************************************************************
   *
   * Go through all the sources to see if they are ready for the next send
   *
   *********************************************************************/

   for(count=ZERO; count < number_of_sources; count++) {

      if(processing_list[count].generate == TRUE) {

         /*****************************************************************
         *
         * If we are here then this source is ready for a new set of packets
         * to be sent
         *
         *****************************************************************/

         struct messagebuffer     sendbuffer;
         struct generator_buffer  *message_to_generator;
         int                      sendsuccess  = ZERO;
         message_to_generator     = (struct generator_buffer*)
                                    (sendbuffer.text);

         message_to_generator->generator_cont  = PLZCONTINUE;
         message_to_generator->source          = source_list[count].address;
         message_to_generator->destination     =
                                       processing_list[count].destination;
         message_to_generator->hop_limit       =
            processing_list[count].hop_count + source_list[count].hop_count;
         message_to_generator->protocol        =
                                       SOURCEROUTE + IPPROTO_ICMPV6;
         /*****************************************************************
         *
         * Assign a random id number to this packet
         * NOTE: if troubleshoot a sequential number is assigned to allow
         *        scripts to run
         *       and sequence is always 1
         *
         *****************************************************************/

         if (TROUBLESHOOT) {
```

```
   processing_list[count].id= probe_number_counter;
   probe_number_counter = probe_number_counter + 1;
   processing_list[count].sequence       = 1;

} else {

   processing_list[count].id             =
                               rand() % (HIGH - LOW + 1) + LOW;
   processing_list[count].sequence       =
                               rand() % (HIGH - LOW + 1) + LOW;

}//if (TROUBLESHOOT) {
message_to_generator->port           = processing_list[count].id;
message_to_generator->sequence       =
                                processing_list[count].sequence;

if (TROUBLESHOOT==2) {

   fprintf(stderr,"About to send the message to the generator:\n");
   fprintf(stderr, "The value of generator_cont = %i\n", \
                            message_to_generator->generator_cont);
   char  temp_addr_name[INET6_ADDRSTRLEN+1];
   inet_ntop(AF_INET6, &message_to_generator->source,
                               temp_addr_name, INET6_ADDRSTRLEN);
   fprintf(stderr,"The source address = %s\n", temp_addr_name);
   inet_ntop(AF_INET6, &message_to_generator->destination,
                               temp_addr_name, INET6_ADDRSTRLEN);
   fprintf(stderr, "The destination address = %s\n", \
                                                  temp_addr_name);
   fprintf(stderr,"The hop limit = %i\n", \
                                message_to_generator->hop_limit);
   fprintf(stderr,"The port number = %i\n", \
                                     message_to_generator->port);
   fprintf(stderr,"The sequence number = %i\n", \
                                  message_to_generator->sequence);

}//if (TROUBLESHOOT==2) {

/*****************************************************************
*
* Send the message
*
*****************************************************************/

sendbuffer.type = GENERATOR;

sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                     sizeof(struct generator_buffer), IPC_NOWAIT);

if(sendsuccess != ZERO){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: Message"\
            " send to generator failed.  Error: ");
   print_IPC_error("PROBE MAIN");

} else {

   /**************************************************************
   *
   * Make sure we don't generate another packet until we are told
   * to and set the anonymous
   * clock to now
   *
   **************************************************************/
   processing_list[count].generate = FALSE;
   processing_list[count].time_sent = time(NULL);
   char  temp_addr_name[INET6_ADDRSTRLEN+1];
   inet_ntop(AF_INET6, &(processing_list[count].destination),
```

```
                    temp_addr_name, INET6_ADDRSTRLEN);
            snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: Message sent "\
                     "to probe generator for dest %s from source %i", \
                     temp_addr_name, count);
            call_sys_log(textbuffer);

         }//if(sendsuccess != ZERO){

      }//if(processing_list[count].generate == TRUE) {

   }//for(count=ZERO; count < number_of_sources; count++) {

}//void generate_probes(struct processing_source processing_list[]) {

int  done_check(struct processing_source processing_list[]) {

   /*********************************************************************
   *
   * Check to see if all the sources do not have any remaining destinations
   * if they do not then we are done probing
   *
   *********************************************************************/

   int count = ZERO;

   for(count=ZERO; count < number_of_sources; count++) {

      if(cmpaddr(&empty, &(processing_list[count].destination), 128) !=
         ZERO) {

         return PLZCONTINUE;

      }//if(cmpaddr(&empty, &(processing_list[count].destination), 128) != Z

   }//for(count=ZERO; count < number_of_sources; count++) {

   return STOP;


}//int  done_check(struct processing_source processing_list[]) {

int  check_pointer() {

   /*********************************************************************
   *
   * Checks to see if it is time to check point our progress
   *
   *********************************************************************/

   time_t curtime = time(NULL);
   if ((curtime - check_point_time) > CHECK_POINT_TIME) {

      int temp = ckpt();
      if (temp != ZERO) {

         return STOP;

      } else {

         check_point_time = curtime;
         return PLZCONTINUE;

      }//

   } else {

      return PLZCONTINUE;

   }//if ((curtime - check_point_time) > CHECK_POINT_TIME) {
```

```
}//int  check_pointer()

void process_packet(struct processing_source processing_list[]){

   struct messagebuffer     recvbuffer, sendbuffer;
   //Buffer for messages from receive  function
   struct receive_buffer    *message_from_receiver;
   int                      done         = PLZCONTINUE;
   int                      rcvsuccess   = ZERO;
   int                      sendsuccess  = ZERO;
   int                      probe_source = ZERO;

   /*************************************************************************
   *
   * Process responses until the queue is empty
   *
   *************************************************************************/

   while (done == PLZCONTINUE){

      //Get the message off the queue, do not block
      rcvsuccess = msgrcv(queueid, (void *) &recvbuffer,
                       sizeof(struct receive_buffer), RECEIVE, IPC_NOWAIT);

      if (TROUBLESHOOT==3) {

         fprintf(stderr, "rcvsuccess = %i\n", rcvsuccess);
         if (rcvsuccess == IPC_ERROR) {

            fprintf(stderr, "errno = %i\n", errno);

         }//if (recvsuccess == IPC_ERROR) {

      }//if (TROUBLESHOOT==3) {


      if(rcvsuccess == IPC_ERROR  && errno == ENOMSG) {

         done = STOP;  // Queue is empty exit loop

      }  else if (rcvsuccess == IPC_ERROR && errno != ENOMSG) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: Message "\
                                          "Receive failed.");
         call_sys_log(textbuffer);
         print_IPC_error("PROBE MAIN");

      } else if (rcvsuccess != IPC_ERROR) {

         message_from_receiver = (struct receive_buffer*) (recvbuffer.text);

         if (TROUBLESHOOT==3) {

            fprintf(stderr, "PROCESS PACKET: The message received is:\n");
            char  temp_addr_name[INET6_ADDRSTRLEN+1];
            inet_ntop(AF_INET6, &message_from_receiver->source,
                      temp_addr_name, INET6_ADDRSTRLEN);
            fprintf(stderr,"PROCESS PACKET: The received source address = "\
                           "%s\n", temp_addr_name);
            inet_ntop(AF_INET6, &message_from_receiver->destination,
                      temp_addr_name, INET6_ADDRSTRLEN);
            fprintf(stderr, "PROCESS PACKET: The received destination "\
                            "address = %s\n", temp_addr_name);
            fprintf(stderr,"PROCESS PACKET: The received protocol = %i\n",\
                            message_from_receiver->protocol);
            inet_ntop(AF_INET6, &message_from_receiver->original_source,
                      temp_addr_name, INET6_ADDRSTRLEN);
            fprintf(stderr,"PROCESS PACKET: The received original source"\
```

```
               " address = %s\n", temp_addr_name);
inet_ntop(AF_INET6, &message_from_receiver->original_destination,
          temp_addr_name, INET6_ADDRSTRLEN);
 fprintf(stderr,"PROCESS PACKET: The received original "\
               "destination address = %s\n", temp_addr_name);
 if (message_from_receiver->original_protocol == IPPROTO_ICMPV6){

    fprintf(stderr,"PROCESS PACKET: The received original "\
                  "protocol was ICMPv6\n");

 } else if (message_from_receiver->original_protocol ==
             IPPROTO_UDP) {

    fprintf(stderr,"PROCESS PACKET: The received original "\
                  "protocol was UDP\n");

 } else {

    fprintf(stderr,"PROCESS PACKET: The received original "\
                  "protocol did not match it = %i\n", \
                  message_from_receiver->original_protocol);

 }//if (message_from_receiver->original_protocol == IPPROTO_ICMP)
 fprintf(stderr,"PROCESS PACKET: The received original hop "\
               "limit = %i\n", \
               message_from_receiver->original_hop_limit);
 if (message_from_receiver->icmp_type == TIME_EXCEEDED) {

    fprintf(stderr,"PROCESS PACKET: The received icmp type"\
                  " = TIME Exceeded\n");

 } else if (message_from_receiver->icmp_type ==
             DESTINATION_UNREACHABLE) {

    fprintf(stderr,"PROCESS PACKET: The received icmp type = "\
                  "Destination unreachable\n");

 } else {

    fprintf(stderr,"PROCESS PACKET: The received icmp type was"\
                  " unknown it = %i\n", \
                  message_from_receiver->icmp_type);

 }//if (message_from_receiver->icmp_type == TIME_EXCEEDED) {
 fprintf(stderr,"PROCESS PACKET: The received icmp code = %i\n",\
                message_from_receiver->icmp_code);
 fprintf(stderr,"PROCESS PACKET: The received icmp ident = "\
               "%i\n", message_from_receiver->icmp_ident);
 fprintf(stderr,"PROCESS PACKET: The received icmp sequence "\
               "= %i\n", message_from_receiver->icmp_sequence);
 fprintf(stderr,"PROCESS PACKET: The received original source "\
               "port = %i\n", \
               message_from_receiver->original_source_port);
 fprintf(stderr,"PROCESS PACKET: The received original dest "\
               "port = %i\n", \
               message_from_receiver->original_dest_port);
 fprintf(stderr,"PROCESS PACKET: The received original route"\
               " seg left = %i\n", \
               message_from_receiver->original_route_seg_left);
 inet_ntop(AF_INET6,
          &message_from_receiver->original_route_addr,
          temp_addr_name, INET6_ADDRSTRLEN);
 fprintf(stderr,"PROCESS PACKET: The received original route"\
               " address = %s\n", temp_addr_name);
 fprintf(stderr,"PROCESS PACKET: The received original icmp"\
               " type = %i\n", \
               message_from_receiver->original_icmp_type);
 fprintf(stderr,"PROCESS PACKET: The received original icmp"\
               " code = %i\n", \
```

```
                message_from_receiver->original_icmp_code);

}//if (TROUBLESHOOT==3) {

/*****************************************************************
*
* Locate the source of this message
*
*****************************************************************/
probe_source = ZERO;
while((processing_list[probe_source].id !=
       message_from_receiver->icmp_ident ||
       processing_list[probe_source].sequence !=
       message_from_receiver->icmp_sequence ||
      (message_from_receiver->protocol != IPPROTO_ICMPV6)) &&
      probe_source < number_of_sources){

   if (TROUBLESHOOT==3) {

       fprintf(stderr,"PROCESS PACKET: Attempting locate source"\
                      " of this message comparingin source %i\n",\
                      probe_source);
       fprintf(stderr,"PROCESS PACKET: The processing_list"\
                      "[probe_source].id = %i\n", \
                      processing_list[probe_source].id);
       fprintf(stderr,"PROCESS PACKET: The message_from_"\
                      "receiver->icmp_ident = %i\n", \
                      message_from_receiver->icmp_ident);
       fprintf(stderr,"PROCESS PACKET: The processing_list"\
                      "[probe_source].sequence = %i\n", \
                      processing_list[probe_source].sequence);
       fprintf(stderr,"PROCESS PACKET: The message_from_rece"\
                      "iver->icmp_sequence = %i\n", \
                      message_from_receiver->icmp_sequence);
       char  temp_addr_name[INET6_ADDRSTRLEN+1];
       inet_ntop(AF_INET6,
                 &message_from_receiver->original_source,
                 temp_addr_name, INET6_ADDRSTRLEN);
       fprintf(stderr,"PROCESS PACKET: The message_from_re"\
                      "ceiver->original_source = %s\n", \
                      temp_addr_name);
       inet_ntop(AF_INET6,
                 &source_list[probe_source].address,
                 temp_addr_name, INET6_ADDRSTRLEN);
       fprintf(stderr,"PROCESS PACKET: The source_list[probe"\
                      "_source].address = %s\n", temp_addr_name);
       fprintf(stderr,"PROCESS PACKET: The message_from_receiver"\
                      "->protocol = %i\n", \
                      message_from_receiver->protocol);
       fprintf(stderr,"PROCESS PACKET: The IPPROTO_ICMPV6 = "\
                      "%i\n", IPPROTO_ICMPV6);

   }//if (TROUBLESHOOT==3) {

   probe_source++;

}//while(processing_list[probe_source].id != message_from_receiver-


if (probe_source >= number_of_sources) {

   /*************************************************************
   *
   * If we are here the packet is not from the set we are working
   * on now
   *
   *************************************************************/

   char  temp_addr_name[INET6_ADDRSTRLEN+1];
```

```
   inet_ntop(AF_INET6, &message_from_receiver->source,
             temp_addr_name, INET6_ADDRSTRLEN);
   snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: Received "\
            "unknown packet from %s src", temp_addr_name);
   call_sys_log(textbuffer);

} else {


   if ((message_from_receiver->icmp_type ==
        DESTINATION_UNREACHABLE) &&
      (message_from_receiver->icmp_code !=
        PORT_UNREACHABLE)) {

      char  temp_addr_name[INET6_ADDRSTRLEN+1];
      inet_ntop(AF_INET6, &message_from_receiver->source,
                temp_addr_name, INET6_ADDRSTRLEN);
      snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: Received "\
               "packet from %s src for src %i with unreachable"\
               " message", temp_addr_name, probe_source);
      call_sys_log(textbuffer);

      if (TROUBLESHOOT==3) {

         fprintf(stderr,"PROCESS PACKET: Received packet with "\
                        "unreachable message from %s!\n",\
                        temp_addr_name);

      }//if (TROUBLESHOOT==3) {

      processing_list[probe_source].bad                     = TRUE;

   } else {

      /*********************************************************
      *
      * Add the packet to the processing list as a received packet
      *
      *********************************************************/

      if (cmpaddr(&processing_list[probe_source].response_addr,
                  &empty, 128) == ZERO){

         /*******************************************************
         *
         * If we are here then this is the first response and we
         * need to load the address
         * of the response
         *
         *******************************************************/

         processing_list[probe_source].response_addr =
                                    message_from_receiver->source;
         processing_list[probe_source].response_number =
                 processing_list[probe_source].response_number + 1;

      }else if(cmpaddr(
               &processing_list[probe_source].response_addr,
               &message_from_receiver->source, 128) == ZERO){

         /*******************************************************
         *
         * If we are here then this is the not the first and we
         * just increment the count
         *
         *******************************************************/

         processing_list[probe_source].response_number =
                 processing_list[probe_source].response_number + 1;
```

```
                  if ((message_from_receiver->original_route_seg_left >
                       ZERO)  &&
                      (processing_list[probe_source].response_number >=
                       MIN_RESPONSE)){

                     snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: "\
                              "WARNING: Source %i complaining about hops", \
                              probe_source);
                     call_sys_log(textbuffer);

                  }//if ((message_from_receiver->original_route_seg_left > Z


               } else {

                  /*********************************************************
                  *
                  * If we are here then we have received a response from
                  * more than one destination
                  * to the three probes
                  *
                  *********************************************************/

                  char  temp_addr_name[INET6_ADDRSTRLEN+1];
                  char  temp_addr_name2[INET6_ADDRSTRLEN+1];
                  char  temp_addr_name3[INET6_ADDRSTRLEN+1];
                  inet_ntop(AF_INET6,
                            &processing_list[probe_source].destination,
                            temp_addr_name, INET6_ADDRSTRLEN);
                  inet_ntop(AF_INET6, &message_from_receiver->source,
                            temp_addr_name2, INET6_ADDRSTRLEN);
                  inet_ntop(AF_INET6,
                            &processing_list[probe_source].response_addr,
                            temp_addr_name3, INET6_ADDRSTRLEN);
                  snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: Rcvd pkt"\
                           " from src %s & src %s for dest %s, hop count"\
                           " %i", temp_addr_name, temp_addr_name2, \
                           temp_addr_name3, \
                           processing_list[probe_source].hop_count);
                  call_sys_log(textbuffer);

               }//if (cmpaddr(&processing_list[probe_source].response_addr,&

            }//if ((message_from_receiver->icmp_type == DESTINATION_UNREACHA

         }//if (probe_source >= number_of_sources) {

      }//if(rcvsuccess == IPC_ERROR  && errno == ENOMSG) {

   }//while (done == PLZCONTINUE){

}//void process_packet(struct processing_source processing_list[]){

int process_source(struct processing_source processing_list[]){

  int probe_source = ZERO;

  for (probe_source=ZERO;probe_source<number_of_sources;probe_source++) {

     if (TROUBLESHOOT==4) {

        fprintf(stderr,"Probe Source %i has %i responses\n", (probe_source)\
                , processing_list[probe_source].response_number);

     }//if (TROUBLESHOOT==4) {

     /*********************************************************************
      *
```

```
 *  CASE A
 *
 ********************************************************************/

if((processing_list[probe_source].bad == TRUE) ||
   (processing_list[probe_source].hop_count < ZERO) ||
   (processing_list[probe_source].hop_count >= MAX_HOPS)) {

   if ((processing_list[probe_source].hop_count < ZERO) ||
      (processing_list[probe_source].hop_count >= MAX_HOPS)) {

      snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: WARNING: Process"\
               " source %i exceeded hop limits with hop limit %i", \
               probe_source, processing_list[probe_source].hop_count);
      call_sys_log(textbuffer);

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i hop count out of limits "\
                 "hop_count = %i\n", probe_source, \
                 processing_list[probe_source].hop_count);

      }//if (TROUBLESHOOT==4) {

   } else {

      snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: Process source "\
               "%i received an unreachable message", probe_source);
      call_sys_log(textbuffer);

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i received an unreachable "\
                        "message\n", probe_source);

      }//if (TROUBLESHOOT==4) {

   }//if ((processing_list[probe_source].hop_count <= ZERO) ||

   if ((processing_list[probe_source].direction == positive) &&
       (processing_list[probe_source].hop_count >= MAX_HOPS) &&
        (processing_list[probe_source].source_start == FALSE)) {

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i changing direction\n",\
                 (probe_source));

      }//if (TROUBLESHOOT==4) {

      change_direction(processing_list, probe_source);

   } else {

      /*****************************************************************
      *
      * Remove from the list
      *
      *****************************************************************/

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i is pulling this destination"\
                 " off the list\n", (probe_source));
         fprintf(stderr,"Probe Source %i number_of_remaining_probes "\
                 "= %i\n", (probe_source), \
                 source_list[probe_source].remaining_probe_counter);

      }//if (TROUBLESHOOT==4) {
```

```
      pull_from_list(probe_source,
                     processing_list[probe_source].list_number,
                     processing_list[probe_source].destination);

      snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: Process source"\
               " %i pulled this destination off the list", \
               probe_source);
      call_sys_log(textbuffer);


      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i is pulled this destination"\
                        " off the list\n", (probe_source));
         fprintf(stderr,"Probe Source %i number_of_remaining_probes "\
                        "= %i\n", (probe_source),\
                    source_list[probe_source].remaining_probe_counter);

      }//if (TROUBLESHOOT==4) {

      /*****************************************************************
      *
      * Clear everything
      *
      *****************************************************************/

      clear_from_processing_list(processing_list, probe_source);

      /*****************************************************************
      *
      * We are done with this destination recalc
      *
      *****************************************************************/

      if (source_list[probe_source].number_of_values > ZERO) {

         source_list[probe_source].h_value =
                     calc_p_value(source_list[probe_source].hop_length,
                           source_list[probe_source].number_of_values);

      }//if (source_list[probe_source].number_of_values > ZERO) {

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i calculated a new h_value"\
                 " %i\n", (probe_source), \
                 source_list[probe_source].h_value);

      }//if (TROUBLESHOOT==4) {

   }//if(processing_list[probe_source].direction == positive) {

/*********************************************************************
*
* CASE B
*
*********************************************************************/

}else if(processing_list[probe_source].response_number >=
         MIN_RESPONSE){

  /*******************************************************************
  *
  * If we are here we have received enough responses to process this
  * sources probes
  *
  *******************************************************************/
```

```
if (TROUBLESHOOT==4) {

   fprintf(stderr,"Probe Source %i has received enough responses"\
                  ":\n", (probe_source));
   char  temp_addr_name[INET6_ADDRSTRLEN+1];
   inet_ntop(AF_INET6, &(processing_list[probe_source].destination),
             temp_addr_name, INET6_ADDRSTRLEN);
   fprintf(stderr,"Destination address: %s\n", temp_addr_name);
   inet_ntop(AF_INET6,
             &(processing_list[probe_source].response_addr),
             temp_addr_name, INET6_ADDRSTRLEN);
   fprintf(stderr,"Response address: %s\n", temp_addr_name);
   fprintf(stderr,"Direction %i\n", \
           processing_list[probe_source].direction);
   fprintf(stderr,"Hop Count %i\n", \
           processing_list[probe_source].hop_count);
   fprintf(stderr,"The h_value for this source is %i\n", \
           source_list[probe_source].h_value);

}//if (TROUBLESHOOT==4) {

if (cmpaddr(&processing_list[probe_source].previous_node, &empty,
             128) == ZERO){

  /*****************************************************************
  *
  * CASE 1
  *
  *****************************************************************/

  /*****************************************************************
  *
  * If we are here then this is the first probe in this direction
  *
  *****************************************************************/

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i thinks this is the first "\
                     "probe in this direction:\n", (probe_source));

   }//if (TROUBLESHOOT==4) {

  /*****************************************************************
  *
  * CASE 1.1
  *
  *****************************************************************/

   if(cmpaddr(&processing_list[probe_source].destination,
              &processing_list[probe_source].response_addr,
              128)== ZERO) {

     /*************************************************************
     *
     * Boundary case first probe in this direction and we hit the
     * destination, back up
     *
     *************************************************************/
      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i thinks hit the "\
                 "destination\n", (probe_source));

      }//if (TROUBLESHOOT==4) {

      processing_list[probe_source].response_addr           = empty;
      move(processing_list, probe_source, REVERSE);
      processing_list[probe_source].anonymous_response_count = ZERO;
```

```
} else if (cmpaddr(&processing_list[probe_source].response_addr,
          &source_list[probe_source].address, 128)==ZERO){

  /*************************************************************
  *
  * CASE 1.2
  *
  *************************************************************/

  /*************************************************************
  *
  * Boundary case we hit the source
  *
  *************************************************************/

  processing_list[probe_source].source_start  = TRUE;
  processing_list[probe_source].hop_count     = ZERO;
  move(processing_list, probe_source, NO_REVERSE);
  processing_list[probe_source].anonymous_response_count = ZERO;

} else {

  /*************************************************************
  *
  * CASE 1.3
  *
  *************************************************************/

  /*************************************************************
  *
  * normal case of first probe in this direction so we move it to
  * the previous node
  * area and check to make sure we have seen this one before or
  * add to everyone
  * else's list
  *
  *************************************************************/
   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i recognizes first probe" \
                     "\n", (probe_source));

   }//if (TROUBLESHOOT==4) {

   if (add_node(processing_list[probe_source].response_addr) ==
       NOT_FOUND_AND_INSERTED) {

      /*********************************************************
      *
      * If we are here this is a new node and we need to add it
      * to everyone else's list
      *
      *********************************************************/

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i recognizes thinks this"\
                 " is a new node\n", (probe_source));

      }//if (TROUBLESHOOT==4) {

      if (processing_list[probe_source].direction == positive) {

         if(global_stop(
            processing_list[probe_source].response_addr,
            processing_list[probe_source].destination)
             == NOT_FOUND_AND_INSERTED) {
```

```
      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i recognizes "\
                 "thinks this as a new global stop\n",\
                 (probe_source));

      }//if (TROUBLESHOOT==4) {

   }//if(global_stop(processing_list[probe_source].response

} else {

   if(local_stop(
           processing_list[probe_source].response_addr,
           probe_source)
      == NOT_FOUND_AND_INSERTED) {

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i recognizes "\
                 "thinks this as a new local stop\n", \
                 (probe_source));

      }//if (TROUBLESHOOT==4) {

   }//if(local_stop(processing_list[probe_source].response_

}//if (processing_list[probe_source].direction == positive)

if(insert_on_list(
     processing_list[probe_source].response_addr,
     probe_source) == STOP) {

   return STOP;

}//if(insert_on_list(processing_list[probe_source].response

if (TROUBLESHOOT==4) {

   fprintf(stderr,"Probe Source %i inserted node on the"\
           " list of others\n", (probe_source));

}//if (TROUBLESHOOT==4) {

/*********************************************************
*
* We located a new node so we need to adjust our initial
* h_value
*
*********************************************************/

int temp = processing_list[probe_source].hop_count;
if ((temp >= ZERO) && (temp < MAX_HOPS)){

   source_list[probe_source].hop_length[temp]++;
   source_list[probe_source].number_of_values++;

} else {

   snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: "\
            "ERROR: Process src for src %i calc bad "\
            "val to put in H list, val = %i", \
            probe_source, temp);
   call_sys_log(textbuffer);

}//if ((temp >= ZERO) && (temp < MAX_HOPS)){

if (TROUBLESHOOT==4) {
```

```
            fprintf(stderr,"Probe Source %i added a new value to"\
                           " the h list\n", (probe_source));

         }//if (TROUBLESHOOT==4) {

      }//if (add_node(processing_list[probe_source].response_addr ==

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i putting ther node in "\
                 "previous and changing hop count\n", \
                 (probe_source));

      }//if (TROUBLESHOOT==4) {

      move(processing_list, probe_source, NO_REVERSE);
      processing_list[probe_source].anonymous_response_count = ZERO;

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i changed hop count to "\
                 "%i\n", (probe_source), \
                 processing_list[probe_source].hop_count);

      }//if (TROUBLESHOOT==4) {


   }//if(cmp(&processing_list[probe_source].destination,&processing_

} else if (cmpaddr(&processing_list[probe_source].destination,
           &processing_list[probe_source].response_addr,
           128)== ZERO){

   /*************************************************************
   *
   * CASE 2
   *
   *************************************************************/
   /*************************************************************
   *
   * If we are here then we have arrived at the destination
   * No need to check for stop we are here
   *
   *************************************************************/

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i thinks it is at the "\
                     "destination\n", (probe_source));

   }//if (TROUBLESHOOT==4) {

   if (add_edge(processing_list[probe_source].previous_node,
                processing_list[probe_source].response_addr,
                source_list[probe_source].address,
                processing_list[probe_source].destination,
                processing_list[probe_source].hop_count) ==
                NOT_FOUND_AND_INSERTED){

      /*********************************************************
      *
      * If we are here then then one of the nodes was new and we
      * have added the edge
      *
      *********************************************************/

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i recognizes this as a "\
```

```
                    "new edge\n", (probe_source));

   }//if (TROUBLESHOOT==4) {

}//if (add_edge(processing_list[probe_source].previous_node,

/*************************************************************
*
* Change direction or pull from list if we started at the source
*
*************************************************************/

if (processing_list[probe_source].source_start == FALSE) {

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i is changing the "\
                    "direction\n", (probe_source));

   }//if (TROUBLESHOOT==4) {

   change_direction(processing_list, probe_source);

 } else {

    /*********************************************************
    *
    * Remove from the list
    *
    *********************************************************/

    if (TROUBLESHOOT==4) {

       fprintf(stderr,"Probe Source %i is pulling this "\
                   "destination off the list\n", (probe_source));
       fprintf(stderr,"The value of probe_source is %i\n", \
                    probe_source);
       fprintf(stderr,"The value of list number is %i\n",\
                    processing_list[probe_source].list_number);
       char  temp_addr_name[INET6_ADDRSTRLEN+1];
       inet_ntop(AF_INET6,
                 &processing_list[probe_source].destination,
                 temp_addr_name, INET6_ADDRSTRLEN);
       fprintf(stderr,"The address passed in = %s\n", \
                 temp_addr_name);
       fprintf(stderr,"Probe Source %i number_of_re"\
               "maining_probes = %i\n", (probe_source), \
              source_list[probe_source].remaining_probe_counter);

    }//if (TROUBLESHOOT==4) {

    pull_from_list(probe_source,
                   processing_list[probe_source].list_number,
                   processing_list[probe_source].destination);

    if (TROUBLESHOOT==4) {

       fprintf(stderr,"Probe Source %i is pulled this "\
                   "destination off the list\n", (probe_source));
       fprintf(stderr,"Probe Source %i number_of_re"\
               "maining_probes = %i\n", (probe_source),
              source_list[probe_source].remaining_probe_counter);

    }//if (TROUBLESHOOT==4) {

    /*********************************************************
    *
    * Clear everything
    *
```

```
                    ************************************************************/

                    clear_from_processing_list(processing_list, probe_source);

                    /************************************************************
                    *
                    * We are done with this destination recalc
                    *
                    ************************************************************/

                    if (source_list[probe_source].number_of_values > ZERO) {

                       source_list[probe_source].h_value =
                          calc_p_value(source_list[probe_source].hop_length,
                                       source_list[probe_source].number_of_values);

                    }//if (source_list[probe_source].number_of_values > ZERO) {

                    if (TROUBLESHOOT==4) {

                       fprintf(stderr,"Probe Source %i calculated a new "\
                                      "h_value %i\n", (probe_source), \
                                      source_list[probe_source].h_value);

                    }//if (TROUBLESHOOT==4) {

                 }//if (processsing_list[probe_source].source_start == FALSE) {

              } else if (cmpaddr(&processing_list[probe_source].response_addr,
                         &source_list[probe_source].address, 128)==ZERO){
////////case 3

                 /************************************************************
                 *
                 * If we are here then we have arrived at the source
                 *
                 ************************************************************/

                 if (TROUBLESHOOT==4) {

                    fprintf(stderr,"Probe Source %i thinks it is at the source"\
                                   "\n", (probe_source));

                 }//if (TROUBLESHOOT==4) {


                 if (add_edge(processing_list[probe_source].previous_node,
                              processing_list[probe_source].response_addr,
                              source_list[probe_source].address,
                              processing_list[probe_source].destination,
                              processing_list[probe_source].hop_count) ==
                              NOT_FOUND_AND_INSERTED){

                    /*********************************************************
                    *
                    * If we are here then then one of the nodes was new and we
                    * have added the edge
                    *
                    *********************************************************/

                    if (TROUBLESHOOT==4) {

                       fprintf(stderr,"Probe Source %i recognizes this as a "\
                                      "new edge\n", (probe_source));

                    }//if (TROUBLESHOOT==4) {


                 }//if (add_edge(processing_list[probe_source].previous_node,
```

```
/*****************************************************************
*
* Remove from the list
*
*****************************************************************/

if (TROUBLESHOOT==4) {

   fprintf(stderr,"Probe Source %i is pulling this destination "\
                  "off the list\n", (probe_source));
   fprintf(stderr,"The value of probe_source is %i\n", \
                   probe_source);
   fprintf(stderr,"The value of list number is %i\n", \
                   processing_list[probe_source].list_number);
   char  temp_addr_name[INET6_ADDRSTRLEN+1];
   inet_ntop(AF_INET6,
             &processing_list[probe_source].destination,
             temp_addr_name, INET6_ADDRSTRLEN);
   fprintf(stderr,"The address passed in = %s\n",temp_addr_name);
   fprintf(stderr,"Probe Source %i number_of_remaining_probes "\
           "= %i\n", (probe_source), \
           source_list[probe_source].remaining_probe_counter);

}//if (TROUBLESHOOT==4) {

pull_from_list(probe_source,
               processing_list[probe_source].list_number,
               processing_list[probe_source].destination);

if (TROUBLESHOOT==4) {

   fprintf(stderr,"Probe Source %i is pulled this destination "\
                  "off the list\n", (probe_source));
   fprintf(stderr,"Probe Source %i number_of_remaining_probes "\
                  "= %i\n", (probe_source), \
              source_list[probe_source].remaining_probe_counter);

}//if (TROUBLESHOOT==4) {

/*****************************************************************
*
* Clear everything
*
*****************************************************************/

clear_from_processing_list(processing_list, probe_source);

/*****************************************************************
*
* We are done with this destination recalc
*
*****************************************************************/

if (source_list[probe_source].number_of_values > ZERO) {

   source_list[probe_source].h_value =
          calc_p_value(source_list[probe_source].hop_length,
                       source_list[probe_source].number_of_values);

}//if (source_list[probe_source].number_of_values > ZERO) {

if (TROUBLESHOOT==4) {

   fprintf(stderr,"Probe Source %i calculated a new h_value "\
           "%i\n", (probe_source), \
           source_list[probe_source].h_value);

}//if (TROUBLESHOOT==4) {
```

```
          } else if (processing_list[probe_source].direction == positive) {
//////////case 4
             /*************************************************************
             *
             * If we are here then we have move in a positive direction
             *
             *************************************************************/

             if (TROUBLESHOOT==4) {

                fprintf(stderr,"Probe Source %i thinks it just moved in the"\
                               " positive direction\n", (probe_source));

             }//if (TROUBLESHOOT==4) {

             if (add_edge(processing_list[probe_source].previous_node,
                          processing_list[probe_source].response_addr,
                          source_list[probe_source].address,
                          processing_list[probe_source].destination,
                          processing_list[probe_source].hop_count) ==
                          NOT_FOUND_AND_INSERTED){

                /*********************************************************
                *
                * If we are here then one of the nodes was new and we have
                * added the edge
                *
                *********************************************************/

                if (TROUBLESHOOT==4) {

                   fprintf(stderr,"Probe Source %i recognizes this as a new"\
                                  " edge\n", (probe_source));

                }//if (TROUBLESHOOT==4) {

                if (add_node(processing_list[probe_source].response_addr) ==
                    NOT_FOUND_AND_INSERTED) {

                   /******************************************************
                   *
                   * If we are here this is a new node and we need to add it
                   * to everyone else's list
                   *
                   ******************************************************/

                   if (TROUBLESHOOT==4) {

                      fprintf(stderr,"Probe Source %i recognizes this as a "\
                              "new node and is adding to everyone else\n", \
                              (probe_source));

                   }//if (TROUBLESHOOT==4) {

                   if(insert_on_list(processing_list[probe_source].response_addr,
                                     probe_source) == STOP) {

                      return STOP;

                   }//if(insert_on_list(processing_list[probe_source].response

                   if (TROUBLESHOOT==4) {

                      fprintf(stderr,"Probe Source %i has added this to "\
                                     "everyone else\n", (probe_source));

                   }//if (TROUBLESHOOT==4) {
```

```
      /*********************************************************
      *
      * We located a new node so we need to add this to the
      * h list
      *
      *********************************************************/

      int temp = processing_list[probe_source].hop_count;
      if ((temp >= ZERO) && (temp < MAX_HOPS)){

         source_list[probe_source].hop_length[temp]++;
         source_list[probe_source].number_of_values++;

      } else {

         snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: ERROR:"\
                  " Process src for src %i calc bad val to put"\
                  " in H list, val = %i", probe_source, temp);
         call_sys_log(textbuffer);

      }//if ((temp >= ZERO) && (temp < MAX_HOPS)){

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i adjusted h_value to:"\
                 " %i\n", (probe_source),\
                 source_list[probe_source].h_value);

      }//if (TROUBLESHOOT==4) {

   }//if (add_node(processing_list[probe_source].response_addr ==

}//if (add_edge(processing_list[probe_source].previous_node,

/*****************************************************************
*
* Now we need to decide what to do
*
*****************************************************************/

if(global_stop(processing_list[probe_source].response_addr,
               processing_list[probe_source].destination)
                == NOT_FOUND_AND_INSERTED) {

   /*************************************************************
   *
   * If we are here then this is a new global edge move forward
   *
   *************************************************************/

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i recognizes this as a new"\
                     " global edge\n", (probe_source));

   }//if (TROUBLESHOOT==4) {

   move(processing_list, probe_source, NO_REVERSE);
   processing_list[probe_source].anonymous_response_count = ZERO;

} else {

   /*************************************************************
   *
   * We have seen this node before, change direction or stop
   *
   *************************************************************/

   if (processing_list[probe_source].source_start == FALSE) {
```

```
   change_direction(processing_list, probe_source);

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i is changing "\
                     "direction\n", probe_source);

   }//if (TROUBLESHOOT==4) {

} else {

   /*********************************************************
   *
   * Remove from the list
   *
   *********************************************************/

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i is pulling this "\
              "destination off the list\n", (probe_source));
      fprintf(stderr,"Probe Source %i number_of_remaining_"\
              "probes = %i\n", (probe_source), \
              source_list[probe_source].remaining_probe_counter);

   }//if (TROUBLESHOOT==4) {

   pull_from_list(probe_source,
                  processing_list[probe_source].list_number,
                  processing_list[probe_source].destination);

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i is pulled this "\
                     "destination off the list\n", \
                     (probe_source));
      fprintf(stderr,"Probe Source %i number_of_remaining_"\
                     "probes = %i\n", (probe_source), \
           source_list[probe_source].remaining_probe_counter);

   }//if (TROUBLESHOOT==4) {

   /*********************************************************
   *
   * Clear everything
   *
   *********************************************************/

   clear_from_processing_list(processing_list, probe_source);

   /*********************************************************
   *
   * We are done with this destination recalc
   *
   *********************************************************/

   if (source_list[probe_source].number_of_values > ZERO) {

      source_list[probe_source].h_value =
            calc_p_value(source_list[probe_source].hop_length,
                  source_list[probe_source].number_of_values);

   }//if (source_list[probe_source].number_of_values > ZERO) {

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i calculated a new "\
              "h_value %i\n", (probe_source), \
```

```
                source_list[probe_source].h_value);

         }//if (TROUBLESHOOT==4) {

      }//if (processing_list[probe_source].source_start == FALSE) {

   }//if(global_stop(processing_list[probe_source].response_addr, pr

} else {

   /*************************************************************
   *
   * CASE 5
   *
   *************************************************************/

   /*************************************************************
   *
   * If we are here then we have moved in a negative direction
   *
   *************************************************************/
   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i thinks it just moved in the"\
                     " negative direction\n", (probe_source));

   }//if (TROUBLESHOOT==4) {

   if (add_edge(processing_list[probe_source].previous_node,
               processing_list[probe_source].response_addr,
               source_list[probe_source].address,
               processing_list[probe_source].destination,
               processing_list[probe_source].hop_count) ==
               NOT_FOUND_AND_INSERTED){

      /*********************************************************
      *
      * If we are here then one of the nodes was new and we have
      * added the edge
      *
      *********************************************************/

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i recognizes this as a "\
                        "new edge\n", (probe_source));

      }//if (TROUBLESHOOT==4) {

      if (add_node(processing_list[probe_source].response_addr) ==
                  NOT_FOUND_AND_INSERTED) {

         /******************************************************
         *
         * If we are here this is a new node and we need to add it
         * to everyone else's list
         *
         ******************************************************/

         if (TROUBLESHOOT==4) {

            fprintf(stderr,"Probe Source %i recognizes this as a "\
                    "new node and is adding to everyone else\n", \
                    (probe_source));

         }//if (TROUBLESHOOT==4) {

         if(insert_on_list(
                processing_list[probe_source].response_addr,
```

```
            probe_source) == STOP) {

         return STOP;

      }//if(insert_on_list(processing_list[probe_source].response

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i has added this to "\
                        "everyone else\n", (probe_source));

      }//if (TROUBLESHOOT==4) {

      /*********************************************************
      *
      * We located a new node so we need to adjust our initial
      * h_value
      *
      *********************************************************/

      int temp = processing_list[probe_source].hop_count;
      if ((temp >= ZERO) && (temp < MAX_HOPS)){

         source_list[probe_source].hop_length[temp]++;
         source_list[probe_source].number_of_values++;

      } else {

         snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: ERROR:"\
                  " Process src for src %i calc bad val to put"\
                  " in H list, val = %i", probe_source, temp);
         call_sys_log(textbuffer);

      }//if ((temp >= ZERO) && (temp < MAX_HOPS)){

   }//if (add_node(processing_list[probe_source].response_addr ==

}//if (add_edge(processing_list[probe_source].previous_node,

/*****************************************************************
*
* Now we need to decide what to do
*
*****************************************************************/

if((local_stop(processing_list[probe_source].response_addr,
   probe_source) == NOT_FOUND_AND_INSERTED) &&
  (processing_list[probe_source].hop_count > ZERO)){

   /*************************************************************
   *
   * If we are here then this is a new local edge press on
   *
   *************************************************************/

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i recognizes this as a new"\
                     " local edge\n", (probe_source));

   }//if (TROUBLESHOOT==4) {

   move(processing_list, probe_source, NO_REVERSE);
   processing_list[probe_source].anonymous_response_count = ZERO;

} else {

   /*************************************************************
   *
```

```
         * We have seen this edge before, clear everything and get a
         * new node
         *
         ************************************************************/
         /************************************************************
         *
         * Remove from the list
         *
         ************************************************************/

         if (TROUBLESHOOT==4) {

            fprintf(stderr,"Probe Source %i is pulling this "\
                    "destination off the list\n", (probe_source));
           fprintf(stderr,"Probe Source %i number_of_remaining_probes"\
                    " = %i\n", (probe_source), \
                    source_list[probe_source].remaining_probe_counter);

         }//if (TROUBLESHOOT==4) {

         pull_from_list(probe_source,
                        processing_list[probe_source].list_number,
                        processing_list[probe_source].destination);

         if (TROUBLESHOOT==4) {

            fprintf(stderr,"Probe Source %i is pulled this "\
                    "destination off the list\n", (probe_source));
            fprintf(stderr,"Probe Source %i number_of_"\
                    "remaining_probes = %i\n", (probe_source), \
                    source_list[probe_source].remaining_probe_counter);

         }//if (TROUBLESHOOT==4) {

         /************************************************************
         *
         * Clear everything
         *
         ************************************************************/

         clear_from_processing_list(processing_list, probe_source);

         /************************************************************
         *
         * We are done with this destination recalc
         *
         ************************************************************/

         if (source_list[probe_source].number_of_values > ZERO) {

            source_list[probe_source].h_value =
               calc_p_value(source_list[probe_source].hop_length,
                            source_list[probe_source].number_of_values);

         }//if (source_list[probe_source].number_of_values > ZERO) {

         if (TROUBLESHOOT==4) {

            fprintf(stderr,"Probe Source %i calculated a new h_value:"\
                    " %i\n", (probe_source), \
                    source_list[probe_source].h_value);

         }//if (TROUBLESHOOT==4) {

      }//if(local_stop(processing_list[probe_source].response_addr, pro

   }//if (cmp(&processing_list[probe_source].previous_node, &empty, 128

} else if(((time(NULL) - processing_list[probe_source].time_sent) >=
```

```
        WAIT_TIME) &&
       (cmpaddr(&processing_list[probe_source].destination,
               &empty, 128) != ZERO)) {

/*****************************************************************
*
* CASE C
*
*****************************************************************/
/*****************************************************************
*
* If we are here then we have timed out and need to mark this hop
* anonymous
*
*****************************************************************/

if (TROUBLESHOOT==4) {

   fprintf(stderr,"Probe Source %i has timed out:\n", probe_source);
   char  temp_addr_name[INET6_ADDRSTRLEN+1];
   inet_ntop(AF_INET6, &(processing_list[probe_source].destination),
             temp_addr_name, INET6_ADDRSTRLEN);
   fprintf(stderr,"Destination address: %s\n", temp_addr_name);
   inet_ntop(AF_INET6,&(processing_list[probe_source].response_addr),
             temp_addr_name, INET6_ADDRSTRLEN);
   fprintf(stderr,"Response address: %s\n", temp_addr_name);
   fprintf(stderr,"Direction %i\n", \
           processing_list[probe_source].direction);
   fprintf(stderr,"Hop Count %i\n", \
           processing_list[probe_source].hop_count);
   fprintf(stderr,"The time is %u and the processing time is %u\n",\
            time(NULL), processing_list[probe_source].time_sent);
   fprintf(stderr,"The amount of responses received is %i\n",\
                  processing_list[probe_source].response_number);

}//if (TROUBLESHOOT==4) {

if (cmpaddr(&processing_list[probe_source].previous_node, &empty,
    128) == ZERO){

   /*****************************************************************
   *
   * If we are here then this is the first probe in this direction
   * We can't have the first node be anonymous so move the hop count
   * towards source
   *
   *****************************************************************/

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i thinks first node is "\
                     "anonymous, moving hop count\n", probe_source);
      fprintf(stderr,"Probe Source %i hop count is %i\n", \
                     (probe_source), \
                     processing_list[probe_source].hop_count);

   }//if (TROUBLESHOOT==4) {

   if (processing_list[probe_source].hop_count <= ZERO) {

      processing_list[probe_source].response_addr            =
                                 source_list[probe_source].address;
      processing_list[probe_source].source_start             = TRUE;
      processing_list[probe_source].hop_count                = ZERO;
      processing_list[probe_source].anonymous_response_count = ZERO;
      move(processing_list, probe_source, NO_REVERSE);

   } else {
```

```
      processing_list[probe_source].response_addr           = empty;
      move(processing_list, probe_source, REVERSE);
      processing_list[probe_source].anonymous_response_count = ZERO;

   }//if (processing_list[probe_source].hop_count <= ZERO) {

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i moved hop count to %i\n", \
             probe_source, processing_list[probe_source].hop_count);

   }//if (TROUBLESHOOT==4) {

} else {

   /*************************************************************
   *
   * If we are here then this was not the first probe in this
   * direction
   *
   *************************************************************/

   if (processing_list[probe_source].anonymous_response_count <
       MAX_ANONYMOUS) {

     processing_list[probe_source].response_addr = anonymous_address;
      if (add_edge(
           processing_list[probe_source].previous_node,
           processing_list[probe_source].response_addr,
                        source_list[probe_source].address,
                        processing_list[probe_source].destination,
                        processing_list[probe_source].hop_count) ==
                        FOUND){

         /*********************************************************
         *
         * If we are here then we have an error in the engine, there
         * should be no duplicate
         * edges when you have an anonymous node
         *
         *********************************************************/

         char  temp_addr_name[INET6_ADDRSTRLEN+1];
         inet_ntop(AF_INET6, &anonymous_address , temp_addr_name,
                   INET6_ADDRSTRLEN);
         snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: ERROR: "\
                  "Process source had error inserting edge with "\
                  "%s as address", temp_addr_name);
         call_sys_log(textbuffer);

         if (TROUBLESHOOT==4) {

            fprintf(stderr,"ERROR: Probe Source %i recognizes this"\
                           " as a new edge\n", (probe_source));

         }//if (TROUBLESHOOT==4) {

      } else {

         /*********************************************************
         *
         * Increment the address for the next anonymous node
         *
         *********************************************************/

         anonymous_number++;
         anonymous_address.in6_u.u6_addr32[3] =
                                          htonl(anonymous_number);
```

```
   }//if (add_edge(processing_list[probe_source]->previous_node,

}//if (processing_list[probe_source].anonymous_response_count < M

if ((processing_list[probe_source].anonymous_response_count <
     MAX_ANONYMOUS) &&
    (((processing_list[probe_source].hop_count >= ZERO) &&
      (processing_list[probe_source].direction == positive)) ||
     ((processing_list[probe_source].hop_count > ZERO) &&
      (processing_list[probe_source].direction == negative)))) {

   /*************************************************************
   *
   * If we are here then push on
   *
   *************************************************************/

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i is pushing on\n", \
                     (probe_source));

   }//if (TROUBLESHOOT==4) {

   move(processing_list, probe_source, NO_REVERSE);
   processing_list[probe_source].anonymous_response_count++;

   if (TROUBLESHOOT==4) {

      fprintf(stderr,"Probe Source %i is incrementing anonymous"\
                     " to %u\n", (probe_source), \
                     anonymous_address.in6_u.u6_addr32[3]);

   }//if (TROUBLESHOOT==4) {

} else {

   /*************************************************************
   *
   * If we are here we have gone more than the allowed amount of
   * anonymous nodes
   * change direction
   *
   *************************************************************/

   if ((processing_list[probe_source].direction == positive) &&
       (processing_list[probe_source].source_start == FALSE)) {

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i is changing the "\
                 "direction to negative\n", (probe_source));

      }//if (TROUBLESHOOT==4) {

      change_direction(processing_list, probe_source);

   } else {

      if (TROUBLESHOOT==4) {

         fprintf(stderr,"Probe Source %i is pulling this "\
                 "destination off the list\n", (probe_source));
         fprintf(stderr,"Probe Source %i number_of_remaini"\
                 "ng_probes = %i\n", (probe_source), \
              source_list[probe_source].remaining_probe_counter);

      }//if (TROUBLESHOOT==4) {
```

```
                  pull_from_list(probe_source,
                                 processing_list[probe_source].list_number,
                                 processing_list[probe_source].destination);

                  if (TROUBLESHOOT==4) {

                     fprintf(stderr,"Probe Source %i is pulled this "\
                               "destination off the list\n", (probe_source));
                     fprintf(stderr,"Probe Source %i number_of_remain"\
                                    "ing_probes = %i\n", (probe_source), \
                         source_list[probe_source].remaining_probe_counter);

                  }//if (TROUBLESHOOT==4) {

                  /*********************************************************
                  *
                  * Clear everything
                  *
                  *********************************************************/

                  clear_from_processing_list(processing_list, probe_source);

               }//if (processing_list[probe_source].direction == positive) {

            }//if(processing_list[probe_source].anonymous_response_count < MA

         }//if (cmpaddr(&processing_list[probe_source].previous_node, &empty,

      }//else if((time(NULL) - processing_list[probe_source]->time_sent) >= W

   }//for (probe_source=ZERO;probe_source<number_of_sources;probe_source++) {

   return PLZCONTINUE;

}//int process_source(struct processing_source *processing_list[]){


int insert_on_list(const struct in6_addr address, const int finding_source){

   /************************************************************************
   *
   * Inserts a newly found node in to everyone else's list of nodes to probe
   *
   ************************************************************************/

   int count       = ZERO;
   int alreadyhere = FALSE;

   for(count=ZERO; count<number_of_sources;count++) {

      alreadyhere = FALSE;

      if (count != finding_source &&
          (cmpaddr(&address, &source_list[count].address, 128) != ZERO)) {

         struct remaining_probes *temp_probe_pointer;
         temp_probe_pointer = source_list[count].probe_next;

         if (temp_probe_pointer != NULL) {

            while (temp_probe_pointer->next != NULL) {

              if(cmpaddr(&address, &temp_probe_pointer->address, 128)==
                 ZERO){

                  alreadyhere = TRUE;

               }//if(cmpaddr(&address, &temp_probe_pointer->address, 128){
```

```
            temp_probe_pointer = temp_probe_pointer->next;

         }//while (temp_probe_pointer->next != NULL) {

         /*************************************************************
         *
         * Deal with the last element on the list
         *
         *************************************************************/

         if(cmpaddr(&address, &temp_probe_pointer->address, 128)==ZERO){

            alreadyhere = TRUE;

         }//if(cmpaddr(&address, &temp_probe_pointer->address, 128){

         if (alreadyhere == FALSE) {

            temp_probe_pointer->next = (struct remaining_probes*)
                                  malloc(sizeof(struct remaining_probes));

            if (temp_probe_pointer->next == NULL) {

               snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: "\
                        "Unable to allocate memory for new remaining"\
                        " probe");
               call_sys_log(textbuffer);
               return STOP;

            } else {

               temp_probe_pointer = temp_probe_pointer->next;

            }//if (temp_probe_pointer->next == NULL) {

            temp_probe_pointer->address = address;
            temp_probe_pointer->next    = NULL;
            source_list[count].remaining_probe_counter++;

         }//if (alreadyhere == FALSE) {

      } else {

         source_list[count].probe_next = (struct remaining_probes*)
                                malloc(sizeof(struct remaining_probes));
         temp_probe_pointer = source_list[count].probe_next;
         if (temp_probe_pointer == NULL) {

            snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: "\
                    "Unable to allocate memory for new remaining probe");
            call_sys_log(textbuffer);
            return STOP;

         } else {

            temp_probe_pointer->address = address;
            temp_probe_pointer->next    = NULL;
            source_list[count].remaining_probe_counter++;

         }//if (temp_probe_pointer->next == NULL) {

      }//if (temp_probe_pointer != NULL) {

   }//if (count != finding_source) {

}//for(count=ZERO; count<number_of_sources;count++) {

return PLZCONTINUE;
```

```
}//int insert_on_list(struct in6_addr address, int finding node){


void pull_from_list(int probe_source, int value_to_pull,
                    struct in6_addr our_address) {

   /*********************************************************************
   *
   * Remove a node from the probe list for the source specified
   *
   *********************************************************************/

   if (TROUBLESHOOT==5) {

      fprintf(stderr, "Entered pull from list\n");
      fprintf(stderr, "Value of source_list[probe_source].number_of_re"\
                      "maining_probes = %i\n", \
                      source_list[probe_source].remaining_probe_counter);
      fprintf(stderr, "Value of value_to_pull is %i\n", value_to_pull);
      fprintf(stderr, "Value of probe_source = %i\n", probe_source);
      char  temp_addr_name[INET6_ADDRSTRLEN+1];
      inet_ntop(AF_INET6, &our_address , temp_addr_name, INET6_ADDRSTRLEN);
      fprintf(stderr,"The address passed in = %s\n", temp_addr_name);

   }//if (TROUBLESHOOT==5) {

   struct remaining_probes *temp_probe_pointer;
   struct remaining_probes *previous_probe_pointer;
   temp_probe_pointer     = source_list[probe_source].probe_next;

   if ((value_to_pull > ZERO) &&
      (value_to_pull < source_list[probe_source].remaining_probe_counter)) {

      int temp_count = ZERO;
      for(temp_count=ZERO; temp_count<value_to_pull; temp_count++) {

         previous_probe_pointer = temp_probe_pointer;
         temp_probe_pointer = temp_probe_pointer->next;

      }//for(temp_count=ZERO; temp_count<value_to_pull; temp_count++) {

      if(cmpaddr(&our_address, &temp_probe_pointer->address, 128) == ZERO) {

         if (TROUBLESHOOT==5) {

            fprintf(stderr, "PROBE MAIN: Process source matched the probe "\
                    "list address\n");

         }//if (TROUBLESHOOT==5) {

      } else {

         snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: ERROR: Pull from "\
                 "list source DID NOT match probe list address");
         call_sys_log(textbuffer);

         if (TROUBLESHOOT==5) {

            fprintf(stderr, "%s\n", textbuffer);

         }//if (TROUBLESHOOT==5) {

      }//if(cmpadd(&our_address, &final_address, 128) == ZERO) {

      previous_probe_pointer->next = temp_probe_pointer->next;

   } else {

      if (source_list[probe_source].remaining_probe_counter < ZERO){
```

```
            snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: ERROR: Pull from "\
                     "list told to delete item from empty list");
            call_sys_log(textbuffer);

         } else if (value_to_pull >=
                    source_list[probe_source].remaining_probe_counter) {

            snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: ERROR: Pull "\
                     "from list told to delete item greater than number on list");
            call_sys_log(textbuffer);

         } else {

            if(cmpaddr(&our_address, &temp_probe_pointer->address, 128)
               == ZERO) {

               if (TROUBLESHOOT==5) {

                  fprintf(stderr, "PROBE MAIN: Process source matched the "\
                                  "probe list address\n");

               }//if (TROUBLESHOOT==5) {

            } else {

               snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: ERROR: Pull "\
                        "from list source DID NOT match probe list address");
               call_sys_log(textbuffer);

               if (TROUBLESHOOT==5) {

                  fprintf(stderr, "%s\n", textbuffer);

               }//if (TROUBLESHOOT==5) {

            }//if(cmpadd(&our_address, &final_address, 128) == ZERO) {

            source_list[probe_source].probe_next = temp_probe_pointer->next;

         }//if (source_list[probe_source].number_of_remaining_probes < ZERO){

      }//if ((value_to_pull > ZERO) && (value_to_pull < source_list[probe_sourc

      free(temp_probe_pointer);
      source_list[probe_source].remaining_probe_counter--;

      if (TROUBLESHOOT==5) {

         fprintf(stderr, "Value of source_list[probe_source].remaining_"\
                         "probe_counter = %i\n", \
                         source_list[probe_source].remaining_probe_counter);
         fprintf(stderr, "Value of source_list[probe_source].probe_next ="\
                         " %u\n", source_list[probe_source].probe_next);
         fprintf(stderr, "Exit pull from list\n");

      }//if (TROUBLESHOOT==5) {

   }//void pull_from_list(int probe_source, int value_to_pull, struct in6_addr

   void clear_from_processing_list(struct processing_source processing_list[],
                                   int probe_source){

      /*********************************************************************
      *
      * This is to make the code mode readable, it just clears this destination
      * from the
      * processing list
      *
```

```
   ***************************************************************************/

   processing_list[probe_source].destination              = empty;
   processing_list[probe_source].response_addr            = empty;
   processing_list[probe_source].list_number              = ZERO;
   processing_list[probe_source].direction                = ZERO;
   processing_list[probe_source].id                       = ZERO;
   processing_list[probe_source].sequence                 = ZERO;
   processing_list[probe_source].hop_count                = ZERO;
   processing_list[probe_source].time_sent                = ZERO;
   processing_list[probe_source].response_number          = ZERO;
   processing_list[probe_source].previous_node            = empty;
   processing_list[probe_source].anonymous_response_count = ZERO;
   processing_list[probe_source].generate                 = FALSE;
   processing_list[probe_source].source_start             = FALSE;
   processing_list[probe_source].bad                      = FALSE;

}//void clear_from_processing_list(struct processing_source processing_list[


void move(struct processing_source processing_list[], int probe_source,
          int reverse){

   /***************************************************************************
   *
   * This is to make the code mode readable, it moves the probe in the
   * proper direction
   *
   ***************************************************************************/

   processing_list[probe_source].id                       = ZERO;
   processing_list[probe_source].sequence                 = ZERO;

   if (reverse == NO_REVERSE) {

      if (processing_list[probe_source].direction == positive) {

         processing_list[probe_source].hop_count++;

      } else {

         processing_list[probe_source].hop_count--;

      }//if (processing_list[probe_source].direction == positive) {

   } else {

      processing_list[probe_source].hop_count--;

   }//if (reverse == NO_REVERSE) {
   processing_list[probe_source].time_sent                = ZERO;
   processing_list[probe_source].response_number          = ZERO;
   processing_list[probe_source].previous_node            =
                     processing_list[probe_source].response_addr;
   processing_list[probe_source].response_addr            = empty;
   processing_list[probe_source].generate                 = TRUE;
   processing_list[probe_source].bad                      = FALSE;

}//void move(struct processing_source processing_list[], int probe_source){


void change_direction(struct processing_source processing_list[],
                      int probe_source) {

   /***************************************************************************
   *
   * This is to make the code mode readable, it changes the direction of the
   * probe
   *
```

```
   **********************************************************************/

   processing_list[probe_source].response_addr            = empty;
   processing_list[probe_source].id                       = ZERO;
   processing_list[probe_source].sequence                 = ZERO;
   processing_list[probe_source].direction                = negative;
   processing_list[probe_source].time_sent                = ZERO;
   processing_list[probe_source].response_number          = ZERO;
   processing_list[probe_source].previous_node            = empty;
   processing_list[probe_source].anonymous_response_count = ZERO;
   processing_list[probe_source].generate                 = TRUE;
   processing_list[probe_source].source_start             = FALSE;
   processing_list[probe_source].hop_count                =
                               source_list[probe_source].h_value;
   processing_list[probe_source].bad                      = FALSE;

}//void change_direction(struct processing_source processing_list[], in prob
```

PROBE_MAIN_HELPER.H

```
/*****************************************************************************
 * This is the header file for all the programs that help main run
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_main_helper.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_PROBE_MAIN_HELPER
#define INCLUSION_GUARD_PROGRAM_PROBE_MAIN_HELPER

   int  process_receive(struct processing_source processing_list[]);
// Processes all the received packets
   void fill_list(struct processing_source processing_list[]);
   void generate_probes(struct processing_source processing_list[]);
   int  done_check(struct processing_source processing_list[]);
   int  check_pointer();

#endif //INCLUSION_GUARD_PROGRAM_PROBE_MAIN_HELPER
```

PROBE_MAIN_STUB.C

```
/*****************************************************************************
 * This is the main program of the probing engine and controls all operation
 * (////////////////////////////STUB/////////////////////////////////////)
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_main.c
 *
 *****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <netinet/in.h>
#include <libnet.h>
#include <signal.h>
#include <unistd.h>
#include <wait.h>
#include "../src/probe_main.h"
#include "../src/probe_generator.h"
#include "../src/probe_receive.h"
#include "../src/probe_utils.h"
#include "../src/probe_loader.h"
#include "../src/probe_ckpnt.h"
#include "../src/probe_stop.h"
#include "../src/sys_log.h"
#include <sys/socket.h>
#include <arpa/inet.h>
#include <pcap.h>


#define _GNU_SOURCE
#define TROUBLESHOOT 0

/*****************************************************************************
*
* Define constants
*
*****************************************************************************/

   const  int               IPC_ERROR        = -1;
   const  int               LIBNET_ERROR     = -1;
   const  int               positive         =  1;
   const  int               negative         = -1;

/*****************************************************************************
 *
 * Define externals and globals
 *
 *****************************************************************************/

   // Set by calls to the library functions to define errors encountered
   extern int errno;

   //Buffer for messages to the generator function
   struct generator_buffer  *message_to_generator;
   //Buffer for messages from receive  function
   struct receive_buffer    *message_from_receiver;
   // Pointer to the list of sources
   struct sources           *source_list;
   // Pointer to the list of edges
   struct edges             *edge_list;
   // Pointer to the global stop set
   struct global_stops      *global_stop_list;
   // Pointer to the local stop set
```

```
        void*            *probe_source_stop_list;
   // Pointer to the alias list for alias resolution
   struct alias_nodes       *alias_list;


   struct messagebuffer     sendbuffer, recvbuffer;
   // Flag address stating there is no address
   struct in6_addr          empty;
   // address used to mark anonymous nodes
   struct in6_addr          anonymous_address;
   // The id number of the queue we are using, needs to be global for all
          int               queueid;
   //Our IPv6 address
          char              our_ipv6_addr_name[INET6_ADDRSTRLEN+1];
   //The ipv4 address of the tunnel endpoint
          char              tunnelip[INET_ADDRSTRLEN+1];
   // Loop variable telling us to stop
          int               main_done         = PLZCONTINUE;

   // The total number of sources for our probes
   int  number_of_sources        = ZERO;
   // The total number of edges found
   int  number_of_edges          = ZERO;
   // The number in the global stop table
   int  number_of_global_stops   = ZERO;
   // The number of total nodes in the alias list
   int  number_of_nodes          = ZERO;
   // Used to store the pid of the probe generator
   int  probe_generator_pid      = ZERO;
   // Used to store the pid of the sys log facility
   int  sys_log_pid              = ZERO;
   // Used to store the pid of the probe receiver
   int  probe_receive_pid        = ZERO;
   // Tells us if we are main
   int    is_main                  = TRUE;
   // This records our checkpoint time
   time_t check_point_time         = ZERO;
   // The number we are using in the anonymous nodes
   unsigned int  anonymous_number  = ZERO;
   // The number of packets to be sent for each probe
   int    NUMBER_OF_PACKETS        = ZERO;
   // The minimum number of reesponse we will consider valid
   int    MIN_RESPONSE             = ZERO;
   // The maximum amount of anonymous nodes before we stop probing
   int    MAX_ANONYMOUS            = ZERO;
   // The default value using cdf function to determine our starting hop count
   double DEFAULT_P_VALUE          = 0.0;
   // the time we will wait for a response
   int    WAIT_TIME                = ZERO;
   // How much time should elapse before we check point
   int    CHECK_POINT_TIME         = ZERO;



/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/

 void child_handler(int sig) {

    char textbuffer[MAX_MESSAGE];  // Buffer to hold our message
    int  status;
    int  pid         = ZERO;


    pid = wait(&status);
    if (pid == sys_log_pid) {
```

```
        fprintf(stderr,"PROBE MAIN: Sys log stopped, if this is unplanned "\
                       "check log\n");
        fprintf(stderr,"PROBE MAIN: The sys log process stopped with "\
                       "status: %i\n", status);
        sys_log_pid = ZERO;

    } else if (pid == probe_generator_pid) {

        fprintf(stderr,"PROBE MAIN: Probe Generator stopped, if this is "\
                       "unplanned check log\n");
        snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: The probe generator "\
                       "stopped with status: %i", status);
        call_sys_log(textbuffer);
        probe_generator_pid = ZERO;

    } else if (pid == probe_receive_pid) {

        fprintf(stderr,"PROBE MAIN: Probe Receiver stopped, if this is "\
                       "unplanned check log\n");
        snprintf(textbuffer, MAX_MESSAGE, "PROBE MAIN: The probe receiver "\
                                          "stopped with status: %i", status);
        call_sys_log(textbuffer);
        probe_receive_pid = ZERO;

    } else {

        snprintf(textbuffer, MAX_MESSAGE, "We received a signal but pid did"\
                                          " not match any running process, "\
                                          "status = : %i\n", status);
        call_sys_log(textbuffer);

    }//if (pid == sys_log_pid) {

}//void child_handler(int sig) {

/*****************************************************************************
 *
 * Main function
 *
 *****************************************************************************/

int main(int argc, char *argv[]){

   /*************************************************************************
   *
   * Define variables needed by main
   *
   *************************************************************************/

   struct                   msqid_ds msqid_ds, *buf;

                            buf           = &msqid_ds;
   // Flag to tell if any of the processes should continue or quit
   int                      continue_flag = ZERO;
   // Used by main to store destination addresses
   struct in6_addr          destination;
   // Flag to tell us if we had an error forking
   int                      fork_error    = FALSE;
   // PID returned by fork command
   int                      forked_pid    = ZERO;
   // Hop limit used by main
   int                      hop_limit     = ZERO;
   // Tells us if we are main
   int                      is_main       = TRUE;
   // This will be the value used to determine our queue id
   key_t                    key           = KEY;
   //Initial handle for our packet
   libnet_t                 *packet_handle;
```

```
// Used by main to store payload type, ICMP or UDP
int                     payload_type  = ZERO;
// The value of our receive op!!
int                     rcvsuccess    = ZERO;
// The value of our send operation!!
int                     sendsuccess   = ZERO;
// Used by main to store source addresses
struct in6_addr         source;
// Used by main to store the source port number
int                     source_port   = ZERO;
// Buffer to hold our messages for sys-log
char                    textbuffer[MAX_MESSAGE];


/*************************************************************************
*
* Get our IPv6 address and the tunnel endpoint IPv4 address from startup
* If we did not get them on startup, error out and give usage details
*
*************************************************************************/

if (argc < 3) {

   printf("Probe Engine Program!\n");
   printf("Usage: %s [Tunnel end point ipv4 address] [Your IPv6 address]"\
          "\n",argv[0]);
   printf("Example: %s 172.16.0.1 2200::211:2:0:0:2568\n", argv[0]);
   printf("Thanks for playing!!\n");
   return -1;

 }//if (argc < 3) {

 printf("NOTE: IF YOU ARE NOT ROOT THIS WILL FAIL!!!\n");

 if (TROUBLESHOOT) {

    fprintf(stderr, "I am about to do the strncpy\n");
    fprintf(stderr, "The value of argv[1] = %s\n", argv[1]);
    fprintf(stderr, "The value of argv[2] = %s\n", argv[2]);

 }//if (TROUBLESHOOT) {

 strncpy(tunnelip, argv[1], INET_ADDRSTRLEN);
 strncpy(our_ipv6_addr_name, argv[2], INET6_ADDRSTRLEN);

 if (TROUBLESHOOT) {

   fprintf(stderr, "I am about to do the strncpy\n");

 }//if (TROUBLESHOOT) {

 /*************************************************************************
 *
 * Kick off the signal handler
 *
 *************************************************************************/

 signal(SIGCHLD, child_handler);

 /*************************************************************************
 *
 * Set up the message queue for all to use
 *
 *************************************************************************/

 queueid = msgget(key, 0766 | IPC_CREAT);    // Get our message queue
 if (queueid == IPC_ERROR) {

   fprintf(stderr,"PROBE MAIN: ERROR: Message Get Failed!!!!\n");
```

```
   print_IPC_error("PROBE MAIN");
   return -1;

} else {

   printf("PROBE MAIN:Message get succeeded, Queue =  %i\n", queueid );

}//if (queueid == IPC_ERROR) {

/*************************************************************************
*
* Kick off all the other functions
*
*************************************************************************/

is_main    = TRUE;
fork_error = FALSE;

/*************************************************************************
*
* Kick off syslog
*
*************************************************************************/
forked_pid = fork();

if (forked_pid == ZERO) {

   //We are the child
   is_main = FALSE;
   sys_log(); //We forked successfully we are the child!!!

} else if (forked_pid < ZERO) {

   fprintf(stderr,"PROBE MAIN: ERROR: trying to fork sys log, return = "\
                  "%i\n", forked_pid);
   fork_error = TRUE;

} else {

   //We are main and everything worked
   sys_log_pid = forked_pid;
   forked_pid  = ZERO;

}//if (forked_pid == ZERO) {


/*************************************************************************
*
* Kick off probe receive
*
*************************************************************************/

if (is_main && fork_error == FALSE) {


   forked_pid = fork();

   if (forked_pid == ZERO) {

      //We are the child
      is_main = FALSE;

      probe_receive(); //We forked successfully we are the child!!!

   } else if (forked_pid < ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: trying to "\
                  "fork probe receive, return = %i", forked_pid);
         call_sys_log(textbuffer);
```

```
         fork_error = TRUE;

   } else {

         //We are main and everything worked
         probe_receive_pid = forked_pid;
         if (TROUBLESHOOT == 12){

            fprintf(stderr,"The pid for probe receive is: %i\n",\
                           probe_receive_pid);

         }//if (TROUBLESHOOT == 12){
         forked_pid  = ZERO;

   }//if (forked_pid == ZERO) {

}//   if (is_main) {



/*****************************************************************************
*
* Kick off probe generator
*
*****************************************************************************/

if (is_main && fork_error == FALSE) {


   forked_pid = fork();

   if (forked_pid == ZERO) {

      //We are the child
      is_main = FALSE;
      probe_generator(); //We forked successfully we are the child!!!

   } else if (forked_pid < ZERO) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: trying to "\
                  "fork probe generator, return = %i", forked_pid);
         call_sys_log(textbuffer);
         fork_error = TRUE;

   } else {

         //We are main and everything worked
         probe_generator_pid = forked_pid;
         forked_pid  = ZERO;

   }//if (forked_pid == ZERO) {

}//   if (is_main) {


/*****************************************************************************
*
* Kick off main process  // This will need to be moved to another function
* later
*
*****************************************************************************/

if (is_main && fork_error == FALSE) {

   // The choice the user selected
   int    choice       = ZERO;
   // This causes us to loop until we are done
   int    done         = PLZCONTINUE;
```

```
if ((sys_log_pid == ZERO) || (probe_generator_pid == ZERO) ||
   (probe_receive_pid == ZERO)){


   fprintf(stderr, "We had a bad startup, shutting down!!!\n");
   choice = 2;

}//if ((sys_log_pid == ZERO) || (probe_generator_pid == ZERO) ||

if (choice == ZERO) {

   choice = load_sources();
   if (choice != ZERO) {

      fprintf(stderr, "Error loading sources!\n");

   }//if (choice != ZERO) {


}//if (choice == ZERO) {
if (choice == ZERO) {

   choice = load_edges();
   if (choice != ZERO) {

      fprintf(stderr, "Error loading edges!\n");

   }//if (choice != ZERO) {

}//if (choice == ZERO) {
if (choice == ZERO) {

   choice = load_stop_list();
   if (choice != ZERO) {

      fprintf(stderr, "Error loading stop list!\n");

   }//if (choice != ZERO) {

}//if (choice == ZERO) {
if (choice == ZERO) {

   choice = load_probe_list();
   if (choice != ZERO) {

      fprintf(stderr, "Error loading probe list!\n");

   }//if (choice != ZERO) {

}//if (choice == ZERO) {

if (choice == ZERO) {

   printf("What do you want to do?\n");
   printf("SEND/RECEIVE/Check stops/enter edges          =  1\n");
   printf("QUIT                                          =  2\n");
   printf("Entry                                         =  ");
   scanf ("%d", &choice);
   printf("\n");

} else {

   fprintf(stderr,"PROBE MAIN: ERROR in loading startup files."\
                  "  Shutting down!\n");

}//if (choice == ZERO) {

if (choice == 1) {
```

```
while (done == PLZCONTINUE) {

   printf("What would you like to do?\n");
   printf("SEND                      =  1\n");
   printf("RECEIVE                   =  2\n");
   printf("Stop edge check           =  3\n");
   printf("Enter an edge             =  4\n");
   printf("Enter a node              =  5\n");
   printf("Entry           =  ");
   scanf ("%d", &choice);
   printf("\n");
   if (choice == 1) {

      // Select the desired operation.
      printf("Send a message to Packet Generator\n");
      printf("Continue Flag :\n");
      printf("Continue               =  1\n");
      printf("Stop                   =  2\n");
      printf("Entry                  =  ");
      scanf ("%d", &choice);
      printf("\n");
      continue_flag = choice;
      printf("Source Route?\n");
      printf("Yes              =  1\n");
      printf("No               =  2\n");
      printf("Entry            =  ");
      scanf ("%d", &choice);
      printf("\n");
      payload_type = ZERO;
      if (choice == 1) {

         payload_type = SOURCEROUTE;

      }//if (choice == 1) {
      int check = ZERO;
      char  tempip[INET6_ADDRSTRLEN+1];
      if (payload_type == SOURCEROUTE){

         while (check <= ZERO) {
             printf("Source IPv6 address you want to pass: ");
             scanf("%46s", tempip);
             printf("\n");
             check = inet_pton(AF_INET6, tempip, &source);
             if (check <= ZERO){

                fprintf(stderr,"Error in the address passed in,"\
                               " do again\n");
                fprintf(stderr,"Input string = %s\n", tempip);

             }//if (check != ZERO){
         }// while (check != ZERO) {

      }//if (payload_type == SOURCEROUTE){
      check = ZERO;
      while (check <= ZERO) {
         printf("Destination IPv6 address you want to pass: ");
         scanf("%46s", tempip);
         printf("\n");
         check = inet_pton(AF_INET6, tempip, &destination);
         if (check <= ZERO){

            fprintf(stderr,"Error in the address passed in, do "\
                           "again %i\n", check);
            fprintf(stderr,"Input string = %s\n", tempip);

         } else  if (payload_type == SOURCEROUTE) {

            check = inet_pton(AF_INET6, tempip, &source);
```

```
    }//if (check != ZERO){
}// while (check != ZERO) {

 printf("What hop limit do you want 1 - 128\n");
 scanf ("%d", &choice);
 printf("\n");
 hop_limit = choice;
 printf("Do you want UDP or ICMP\n");
 printf("UDP                  =  1\n");
 printf("ICMP                 =  2\n");
 scanf ("%d", &choice);
 printf("\n");
 if (choice == 1) {

    payload_type += IPPROTO_UDP;
    printf("Please enter the source port number\n");
    scanf ("%d", &choice);
    printf("\n");
    source_port = choice;

 } else {

    payload_type += IPPROTO_ICMPV6;
    printf("Please enter the identifier number\n");
    scanf ("%d", &choice);
    printf("\n");
    source_port = choice;

 }//if (choice == UDP) {

 /*************************************************************
 *
 * Prep the message to send
 *
 *************************************************************/

 message_to_generator =
                   (struct generator_buffer*) (sendbuffer.text);

 message_to_generator->generator_cont  = continue_flag;
 message_to_generator->source          = source;
 message_to_generator->destination     = destination;
 message_to_generator->hop_limit       = hop_limit;
 message_to_generator->protocol        = payload_type;
 message_to_generator->port            = source_port;

 /*************************************************************
 *
 * Send the message
 *
 *************************************************************/

 sendbuffer.type = GENERATOR;

 sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                      sizeof(struct generator_buffer),
                      IPC_NOWAIT);

 if(sendsuccess != ZERO){

    printf("PROBE MAIN: ERROR: Message send failed.  Error: ");
    print_IPC_error("PROBE MAIN");

 } else {

    sendbuffer.type = SYSLOG;
    snprintf(sendbuffer.text, MAX_MESSAGE, "PROBE MAIN: "\
             "Message sent to probe generator");
```

```
      sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                        sizeof(sendbuffer.text), IPC_NOWAIT);
      if(sendsuccess != ZERO){

        print_IPC_error("PROBE MAIN");
        printf("%s\n",sendbuffer.text);

      }//if(sendsuccess != ZERO){

      printf("Message was sent\n");

   }//if(sendsuccess != ZERO){

} else if (choice == 2) {


   //Get the message off the queue, do not block
   rcvsuccess = msgrcv(queueid, (void *) &recvbuffer,
                       sizeof(struct receive_buffer),
                       RECEIVE, IPC_NOWAIT);

   if (TROUBLESHOOT == 15) {

      fprintf(stderr, "rcvsuccess = %i\n", rcvsuccess);
      if (rcvsuccess == IPC_ERROR) {

         fprintf(stderr, "errno = %i\n", errno);

      }//if (recvsuccess == IPC_ERROR) {

   }//if (TROUBLESHOOT == 15) {


  if(rcvsuccess == IPC_ERROR  && errno != ENOMSG) {


     snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: Message"\
                                      " Receive failed.");
     call_sys_log(textbuffer);
     print_IPC_error("PROBE MAIN");
     done = STOP;

  } else if (rcvsuccess == IPC_ERROR  && errno == ENOMSG) {

     printf("No messages from probe receive on the queue\n");

  }  else {

    message_from_receiver =
                 (struct receive_buffer*) (recvbuffer.text);

    printf("The message received is:\n");
    char  temp_addr_name[INET6_ADDRSTRLEN+1];
    inet_ntop(AF_INET6, &message_from_receiver->source,
              temp_addr_name, INET6_ADDRSTRLEN);
    printf("The received source address = %s\n", temp_addr_name);
    inet_ntop(AF_INET6, &message_from_receiver->destination,
              temp_addr_name, INET6_ADDRSTRLEN);
    printf("The received destination address = %s\n", \
            temp_addr_name);
    printf("The received protocol = %i\n", \
            message_from_receiver->protocol);
    inet_ntop(AF_INET6, &message_from_receiver->original_source,
              temp_addr_name, INET6_ADDRSTRLEN);
    printf("The received original source address = %s\n",\
            temp_addr_name);
    inet_ntop(AF_INET6,
              &message_from_receiver->original_destination,
              temp_addr_name, INET6_ADDRSTRLEN);
```

```
    printf("The received original destination address = %s\n", \
           temp_addr_name);
    printf("The received original protocol = %i\n", \
           message_from_receiver->original_protocol);
    printf("The received original hop limit = %i\n", \
           message_from_receiver->original_hop_limit);
    printf("The received icmp type = %i\n", \
           message_from_receiver->icmp_type);
    printf("The received icmp code = %i\n", \
           message_from_receiver->icmp_code);
    printf("The received icmp ident = %i\n", \
           message_from_receiver->icmp_ident);
    printf("The received icmp sequence = %i\n", \
           message_from_receiver->icmp_sequence);
    printf("The received original source port = %i\n", \
           message_from_receiver->original_source_port);
    printf("The received original dest port = %i\n", \
           message_from_receiver->original_dest_port);
    printf("The received original route seg left = %i\n", \
           message_from_receiver->original_route_seg_left);
    inet_ntop(AF_INET6,
              &message_from_receiver->original_route_addr,
              temp_addr_name, INET6_ADDRSTRLEN);
    printf("The received original route address = %s\n",\
            temp_addr_name);
    printf("The received original icmp type = %i\n",\
            message_from_receiver->original_icmp_type);
    printf("The received original icmp code = %i\n", \
            message_from_receiver->original_icmp_code);

  }//if(rcvsuccess == IPC_ERROR  && errno != ENOMSG) {

} else if (choice == 3) {

   printf("Local or global?\n");
   printf("Local      =  1\n");
   printf("Global     =  2\n");
   printf("Entry      =  ");
   scanf ("%d", &choice);
   printf("\n");

   if (choice == 1) {

      printf("Enter the source number, 1 - %i\n", \
             (number_of_sources + 1));
      printf("Entry      =  ");
      scanf ("%d", &choice);
      printf("\n");
      choice--;
      int check = ZERO;
      char  tempip[INET6_ADDRSTRLEN+1];
      while (check <= ZERO) {
         printf("IPv6 address you want to pass to the stop "\
                "check: ");
         scanf("%46s", tempip);
         printf("\n");
         check = inet_pton(AF_INET6, tempip, &source);
         if (check <= ZERO){

            fprintf(stderr,"Error in the address passed in, "\
                           "do again\n");
            fprintf(stderr,"Input string = %s\n", tempip);

         }//if (check != ZERO){
       }// while (check != ZERO) {
      check = local_stop(source, choice);
      if (check == FOUND) {

         printf("local stop returned FOUND\n");
```

```
      } else if (check == NOT_FOUND_AND_INSERTED) {

         printf("local stop returned NOT FOUND AND INSERTED\n");

      } else {

         printf("local stop returned unknown value = %i\n", check);

      }//if (check == FOUND) {

   } else {

      int check = ZERO;
      char  tempip[INET6_ADDRSTRLEN+1];
      while (check <= ZERO) {
         printf("IPv6 address you want to pass to the stop "\
                "check: ");
         scanf("%46s", tempip);
         printf("\n");
         check = inet_pton(AF_INET6, tempip, &source);
         if (check <= ZERO){

            fprintf(stderr,"Error in the address passed in, "\
                           "do again\n");
            fprintf(stderr,"Input string = %s\n", tempip);

         }//if (check != ZERO){
      }// while (check != ZERO) {
      check = ZERO;
      while (check <= ZERO) {
         printf("IPv6 destination address you want to pass to "\
                "the stop check: ");
         scanf("%46s", tempip);
         printf("\n");
         check = inet_pton(AF_INET6, tempip, &destination);
         if (check <= ZERO){

            fprintf(stderr,"Error in the address passed in, "\
                           "do again\n");
            fprintf(stderr,"Input string = %s\n", tempip);

         }//if (check != ZERO){
      }// while (check != ZERO) {
      check = global_stop(source, destination);
      if (check == FOUND) {

         printf("global stop returned FOUND\n");

      } else if (check == NOT_FOUND_AND_INSERTED) {

         printf("global stop returned NOT FOUND AND INSERTED\n");

      } else {

         printf("global stop returned unknown value = %i\n",check);

      }//if (check == FOUND) {

   }//if (choice == 1) {

} else if (choice == 4) {

   int check = ZERO;
   char  tempip[INET6_ADDRSTRLEN+1];
   struct in6_addr edge_v1;
   struct in6_addr edge_v2;
   struct in6_addr starting_source;
   struct in6_addr final_dest;
```

```
int    hop_count;
while (check <= ZERO) {
   printf("IPv6 address for edge v1: ");
   scanf("%46s", tempip);
   printf("\n");
   check = inet_pton(AF_INET6, tempip, &edge_v1);
   if (check <= ZERO){

      fprintf(stderr,"Error in the address passed in, "\
              "do again\n");
      fprintf(stderr,"Input string = %s\n", tempip);

   }//if (check != ZERO){
}// while (check != ZERO) {
check = ZERO;
while (check <= ZERO) {
   printf("IPv6 address for edge v2: ");
   scanf("%46s", tempip);
   printf("\n");
   check = inet_pton(AF_INET6, tempip, &edge_v2);
   if (check <= ZERO){

      fprintf(stderr,"Error in the address passed in, "\
                     "do again\n");
      fprintf(stderr,"Input string = %s\n", tempip);

   }//if (check != ZERO){
}// while (check != ZERO) {
check = ZERO;
while (check <= ZERO) {
   printf("IPv6 address for starting_source: ");
   scanf("%46s", tempip);
   printf("\n");
   check = inet_pton(AF_INET6, tempip, &starting_source);
   if (check <= ZERO){

      fprintf(stderr,"Error in the address passed in, do "\
                     "again\n");
      fprintf(stderr,"Input string = %s\n", tempip);

   }//if (check != ZERO){
}// while (check != ZERO) {
check = ZERO;
while (check <= ZERO) {
   printf("IPv6 address for final_dest: ");
   scanf("%46s", tempip);
   printf("\n");
   check = inet_pton(AF_INET6, tempip, &final_dest);
   if (check <= ZERO){

      fprintf(stderr,"Error in the address passed in, do "\
                     "again\n");
      fprintf(stderr,"Input string = %s\n", tempip);

   }//if (check != ZERO){
}// while (check != ZERO) {
printf("Hop count?\n");
scanf ("%d", &hop_count);
printf("\n");
check = add_edge(edge_v1, edge_v2, starting_source,
                 final_dest,hop_count );
if (check == FOUND) {

   printf("add_edge returned FOUND\n");

} else if (check == NOT_FOUND_AND_INSERTED) {

   printf("add_edge returned NOT FOUND AND INSERTED\n");
```

```
         } else {

            printf("add_edge returned unknown value = %i\n", check);

         }//if (check == FOUND) {

      } else if (choice == 5) {

         int check = ZERO;
         char  tempip[INET6_ADDRSTRLEN+1];
         while (check <= ZERO) {
            printf("IPv6 address to insert: ");
            scanf("%46s", tempip);
            printf("\n");
            check = inet_pton(AF_INET6, tempip, &source);
            if (check <= ZERO){

               fprintf(stderr,"Error in the address passed in, do "\
                              "again\n");
               fprintf(stderr,"Input string = %s\n", tempip);

            }//if (check != ZERO){
         }// while (check != ZERO) {
         check =  add_node(source);

         if (check == FOUND) {

            printf("add node came back with FOUND\n");

         } else if (check == NOT_FOUND_AND_INSERTED) {

            printf("add node came back with NOT FOUND\n");

         } else {

            printf("add node came back with unknown value %i\n");

         }//if (check == FOUND) {

      }//if (choice == 1) {

      printf("Do it again?\n");
      printf("Yes                  =  1\n");
      printf("No                   =  2\n");
      scanf ("%d", &choice);
      printf("\n");
      if (choice == 2){

         done = STOP;

      }//if (choice == 2){


   }//while (done == PLZCONTINUE) {

}//if (choice == 1) {

/*************************************************************************
*
* Check point if desired
*
*************************************************************************/

printf("Do you want to check point before you quit?\n");
printf("Yes      =  1\n");
printf("No       =  2\n");
printf("Entry    =  ");
scanf ("%d", &choice);
printf("\n");
```

```
 if (choice == 1) {

    int temp = ckpt();
    if (temp != ZERO) {

       printf("We had a failure check pointing, check the log"\
              " for error\n");

    } else {

       printf("Success, everything check pointed\n");

    }//if (temp != ZERO) {

 }//if (choice == 1) {

 /*********************************************************************
 *
 * Kill the probe generator if still running
 *
 *********************************************************************/

 if (probe_generator_pid != ZERO) {

    message_to_generator = (struct generator_buffer*) (sendbuffer.text);
    message_to_generator->generator_cont  = STOP;


   /*********************************************************************
   *
   * Send the message
   *
   *********************************************************************/

   sendbuffer.type = GENERATOR;
   sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                        sizeof(struct generator_buffer), IPC_NOWAIT);

   if(sendsuccess != ZERO){

       snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: Message "\
                "send failed to stop generator.  Error: ");
       call_sys_log(textbuffer);
       print_IPC_error("PROBE MAIN");

   } else {

       snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: Message was sent"\
                                        " to terminate generator");
       call_sys_log(textbuffer);

   }//if(sendsuccess != ZERO){

}// if (probe_generator_pid != ZERO) {


/*********************************************************************
*
* Wait for all the probes to return before killing the receiver
* NOTE: Loop used instead of sleep(10) because signals wake up sleep
*
*********************************************************************/

int count = 0;
if (probe_receive_pid != ZERO) {

   fprintf(stderr,"PROBE MAIN: Waiting for 10 seconds for all probes "\
                  "to return before killing probe receive\n");
```

```
   while(count <= 10){

      count++;
      sleep(1);

   }//while(count <= 5){

}//if (probe_receive_pid != ZERO) {

if (probe_receive_pid != ZERO) {

 int temp = kill(probe_receive_pid, SIGKILL);
 if (temp < ZERO){

    snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: ERROR: Message send "\
                                     "failed to stop receiver.  Error: ");
    call_sys_log(textbuffer);
    print_IPC_error("PROBE MAIN");

 }//if (temp < ZERO){

}//if (probe_receive_pid != ZERO) {

/*************************************************************************
*
* Wait for the probe receive to stop
*
*************************************************************************/

count = 0;
while((probe_receive_pid  != ZERO) && (count <= 5)){

   fprintf(stderr,"PROBE MAIN: Waiting for probe receive to finish\n");
   count++;
   sleep(1);

}//while((probe_receive_pid  != ZERO) && (count <= 5)){

/*************************************************************************
*
* Wait for the probe generator to stop
*
*************************************************************************/

count = ZERO;
while((probe_generator_pid  != ZERO) && (count <= 5)){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE MAIN: Waiting for probe "\
                                    "generator to finish");
   printf("%s\n",textbuffer);
   count++;
   sleep(1);

}//while((probe_generator_pid  != ZERO) && (count <= 5)){

/*************************************************************************
*
* Kill the sys log
*
*************************************************************************/
sendbuffer.type = SYSLOG;
snprintf(sendbuffer.text, 5, "QUIT");

sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                     sizeof(sendbuffer.text), IPC_NOWAIT);
if(sendsuccess != ZERO){

   fprintf(stderr,"PROBE MAIN: ERROR: Unable to tell SYS LOG to STOP\n");
   print_IPC_error("PROBE MAIN");
```

```
      }//if(sendsuccess != ZERO){

      /*************************************************************************
      *
      * Wait for the sys log to stop
      *
      *************************************************************************/

      count = 0;
      while((sys_log_pid  != ZERO) && (count <= 5)){

         printf("PROBE MAIN: Waiting for sys log to finish\n");
         count++;
         sleep(1);

      }//while((sys_log_pid  != ZERO) && (count <= 5)){

      //Destroy the queue and clean up
      int destroy = msgctl(queueid, IPC_RMID, buf);
      if(destroy != ZERO){

         fprintf(stderr,"PROBE MAIN: ERROR: Unable to delete the queue, error"\
                        " number = %i\n", destroy);
         print_IPC_error("PROBE MAIN");

      } else {

         fprintf(stderr,"PROBE MAIN: Message QUEUE destroyed\n");

      }//if(destroy != ZERO){

   }//if (temp < ZERO) {

   /*************************************************************************
   *
   * Shutdown syslog and check to make sure all the children are done before
   * exiting!!!
   *
   *************************************************************************/


}//int main(int argc, char *argv[])
```

PROBE_RECEIVE.C

```
/*****************************************************************************
 * This is the packet generator program of the probing engine
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_receive.c
 *
 *****************************************************************************/

#include <string.h>
#include <netinet/ip6.h>
#include <netinet/udp.h>
#include <netinet/icmp6.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h> /* includes net/ethernet.h */
#include <netinet/ether.h>
#include <netinet/ip.h>
#include <string.h>
#include <netinet/tcp.h>
#include "probe_receive.h"
#include "probe_main.h"
#include "probe_utils.h"
#include <pcap.h>

#define TROUBLESHOOT 0              // 1 = sendmessage 2 = handle_ethernet
                                    // 3 = handle_IP
                                    // 4 = handle_IP6  5 = handle_ICMPv6
                                    // 6 = handle_ICMPv6_Unreachable
                                    // 7 = handle_ICMPv6_original
                                    // 8 = handle_UDP_original
                                    // 9 = handle_Routing_original
#define TUNNEL_PROTO 41             // Type of packet for IPv6 tunnel
#define PACKET_CAPTURE_SIZE 0xFFFF  //This is the max size of the packet
                                    //we want pcap to capture
#define NOT_PROMISCUOS 0            //If PCAP should be in promiscous mode

#ifndef ETHER_HDRLEN
#define ETHER_HDRLEN 14
#endif
#define MAX_BUF       65536 //Maximum buffer we can receive

/*****************************************************************************
 *
 * Define global variables needed by the probe_receive
 *
 *****************************************************************************/

    // Buffer to hold our message for syslog
    char                    textbuffer[MAX_MESSAGE];
    // The handle to our pcap session
    pcap_t                  *pcap_handle;
    // Our ipv6 address we started with
    struct in6_addr         our_ipv6_addr;
    // Source address of packet
    struct in6_addr         source;
    // Destination address of the packet
    struct in6_addr         destination;
    // The protocol carried by this packet
    unsigned int            protocol               = ZERO;
    // Original source address of packet
    struct in6_addr         original_source;
    // Original destination address of packet
```

```
    struct in6_addr          original_destination;
    // Original protocol carried by this packet
    unsigned int             original_protocol       = ZERO;
    // The remaining hop limit on packet
    unsigned int             original_hop_limit      = ZERO;
    // The icmp type of this message
    unsigned int             icmp_type               = ZERO;
    // The icmp code of this message
    unsigned int             icmp_code               = ZERO;
    // The id number of this icmp echo reply
    unsigned int             icmp_ident              = ZERO;
    // The sequence number of this icmp echo reply
    unsigned int             icmp_sequence           = ZERO;
    // The original source port of the sent packet
    unsigned int             original_source_port    = ZERO;
    // The original dest port of packet
    unsigned int             original_dest_port      = ZERO;
    // The # of segements left in original packet
    unsigned int             original_route_seg_left = ZERO;
    // The next sgement address in route header
    struct in6_addr          original_route_addr;
    // The value of the original icmp type
    unsigned int             original_icmp_type      = ZERO;
    // The value of the original icmp code
    unsigned int             original_icmp_code      = ZERO;


/*****************************************************************************
 *
 * This is so if main tells us to stop we handle it and die gracefully
 *
 *****************************************************************************/
void receive_handler(int sig) {

   /*****************************************************************************
    *
    * If we figure out how to interupt pcap_loop then change the interupt, for
    * now this serves no function
    *
    *****************************************************************************/

    fprintf(stderr, "PROBE RECIEVE:  Received message to terminate!\n");
    pcap_breakloop(pcap_handle);

}//void message_handler(int sig) {


/*****************************************************************************
 *
 * Function to send messages back to the main program
 *
 *****************************************************************************/

void sendmessage(void) {


   int                      sendsuccess      = ZERO;
   //Buffer for messages to the main function
   struct receive_buffer    *message_to_main;
   struct messagebuffer     sendbuffer;


   message_to_main = (struct receive_buffer*) (sendbuffer.text);
   sendbuffer.type = RECEIVE;

   /*****************************************************************************
   *
   * Prep message and zero our global values for next receive
```

```
*
*****************************************************************************/

message_to_main->source                 = source;
source.in6_u.u6_addr32[0]               = ZERO;
source.in6_u.u6_addr32[1]               = ZERO;
source.in6_u.u6_addr32[2]               = ZERO;
source.in6_u.u6_addr32[3]               = ZERO;
message_to_main->destination            = destination;
destination.in6_u.u6_addr32[0]          = ZERO;
destination.in6_u.u6_addr32[1]          = ZERO;
destination.in6_u.u6_addr32[2]          = ZERO;
destination.in6_u.u6_addr32[3]          = ZERO;
message_to_main->protocol               = protocol;
protocol                                = ZERO;
message_to_main->original_source        = original_source;
original_source.in6_u.u6_addr32[0]      = ZERO;
original_source.in6_u.u6_addr32[1]      = ZERO;
original_source.in6_u.u6_addr32[2]      = ZERO;
original_source.in6_u.u6_addr32[3]      = ZERO;
message_to_main->original_destination   = original_destination;
original_destination.in6_u.u6_addr32[0] = ZERO;
original_destination.in6_u.u6_addr32[1] = ZERO;
original_destination.in6_u.u6_addr32[2] = ZERO;
original_destination.in6_u.u6_addr32[3] = ZERO;
message_to_main->original_protocol      = original_protocol;
original_protocol                       = ZERO;
message_to_main->original_hop_limit     = original_hop_limit;
original_hop_limit                      = ZERO;
message_to_main->icmp_type              = icmp_type;
icmp_type                               = ZERO;
message_to_main->icmp_code              = icmp_code;
icmp_code                               = ZERO;
message_to_main->icmp_ident             = icmp_ident;
icmp_ident                              = ZERO;
message_to_main->icmp_sequence          = icmp_sequence;
icmp_sequence                           = ZERO;
message_to_main->original_source_port   = original_source_port;
original_source_port                    = ZERO;
message_to_main->original_dest_port     = original_dest_port;
original_dest_port                      = ZERO;
message_to_main->original_route_seg_left = original_route_seg_left;
original_route_seg_left                 = ZERO;
message_to_main->original_route_addr    = original_route_addr;
original_route_addr.in6_u.u6_addr32[0]  = ZERO;
original_route_addr.in6_u.u6_addr32[1]  = ZERO;
original_route_addr.in6_u.u6_addr32[2]  = ZERO;
original_route_addr.in6_u.u6_addr32[3]  = ZERO;
message_to_main->original_icmp_type     = original_icmp_type;
original_icmp_type                      = ZERO;
message_to_main->original_icmp_code     = original_icmp_code;
original_icmp_code                      = ZERO;


/*****************************************************************************
*
* Send the message
*
*****************************************************************************/

sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                     sizeof(struct receive_buffer), IPC_NOWAIT);

if(sendsuccess != ZERO){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: ERROR: Message send to"\
                                    " main failed.  Error: ");
   call_sys_log(textbuffer);
   print_IPC_error("PROBE RECEIVE");
```

```
   } else {

      if (TROUBLESHOOT==1) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Message was sent "\
                                          "to main");
         fprintf(stderr, "%s\n", textbuffer);

      }//if (TROUBLESHOOT==1) {

   }//if(sendsuccess != ZERO){


}//void sendmessage() {

/*****************************************************************************
*
* Function prototypes and requred structures to analyze headers!!
* Taken from: http://www.cet.nau.edu/~mc8/Socket/Tutorials/disect2.c
*
*****************************************************************************/


u_int16_t handle_ethernet
        (u_char *args,const struct pcap_pkthdr* pkthdr,const u_char*
        packet);
u_char* handle_IP
        (u_char *args,const struct pcap_pkthdr* pkthdr,const u_char*
        packet);

u_char* handle_IP6
        (u_char *args,const struct pcap_pkthdr* pkthdr,const u_char*
        packet);

u_char* handle_ICMPv6
        (u_char *args,const struct pcap_pkthdr* pkthdr,const u_char*
        packet);

u_char* handle_ICMPv6_Unreachable
        (u_char *args,const struct pcap_pkthdr* pkthdr,const u_char*
        packet);

u_char* handle_ICMPv6_original
        (u_char *args,const struct pcap_pkthdr* pkthdr,const u_char*
        packet);

u_char* handle_UDP_original
        (u_char *args,const struct pcap_pkthdr* pkthdr,const u_char*
        packet);

u_char* handle_Routing_original
        (u_char *args,const struct pcap_pkthdr* pkthdr,const u_char*
        packet);

/*****************************************************************************
 *
 * This code was taken from:
 * http://www.cet.nau.edu/~mc8/Socket/Tutorials/disect2.c
 * They took it from tcpdump.
 * Web site is missing copyright so it ias added here
 *
 * Copyright (c) 1988, 1989, 1990, 1991, 1993, 1994, 1995, 1996
 *      The Regents of the University of California.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that: (1) source code distributions
 * retain the above copyright notice and this paragraph in its entirety, (2)
 * distributions including binary code include the above copyright notice and
```

```
 * this paragraph in its entirety in the documentation or other materials
 * provided with the distribution, and (3) all advertising materials mentioning
 * features or use of this software display the following acknowledgement:
 * ``This product includes software developed by the University of California,
 * Lawrence Berkeley Laboratory and its contributors.'' Neither the name of
 * the University nor the names of its contributors may be used to endorse
 * or promote products derived from this software without specific prior
 * written permission.
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 *
 * Structure of an internet header, naked of options.
 *
 *
 * We declare ip_len and ip_off to be short, rather than u_short
 * pragmatically since otherwise unsigned comparisons can result
 * against negative integers quite easily, and fail in subtle ways.
 ********************************************************************************/
struct my_ip {
        u_int8_t       ip_vhl;        /* header length, version */
#define IP_V(ip)       (((ip)->ip_vhl & 0xf0) >> 4)
#define IP_HL(ip)      ((ip)->ip_vhl & 0x0f)
        u_int8_t       ip_tos;        /* type of service */
        u_int16_t      ip_len;        /* total length */
        u_int16_t      ip_id;         /* identification */
        u_int16_t      ip_off;        /* fragment offset field */
#define IP_DF 0x4000                  /* dont fragment flag */
#define IP_MF 0x2000                  /* more fragments flag */
#define IP_OFFMASK 0x1fff             /* mask for fragmenting bits */
        u_int8_t       ip_ttl;        /* time to live */
        u_int8_t       ip_p;          /* protocol */
        u_int16_t      ip_sum;        /* checksum */
        struct  in_addr ip_src,ip_dst; /* source and dest address */
};


/********************************************************************************
*
* Function to handle the callback and call the proper people!!
* Taken from: http://www.cet.nau.edu/~mc8/Socket/Tutorials/disect2.c
*
********************************************************************************/

/* looking at ethernet headers */
void my_callback(u_char *args,const struct pcap_pkthdr* pkthdr,
                 const u_char* packet)
{
    //This gets the type of the packet in the ethernet frame
    u_int16_t type = handle_ethernet(args,pkthdr,packet);

    if(type == ETHERTYPE_IP){

        handle_IP(args,pkthdr,packet);

    }//if(type == ETHERTYPE_IP){

}//void my_callback(u_char *args,const struct pcap_pkthdr* pkthdr,const u_char*


/********************************************************************************
*
* Function to take apart the ethernet header!!
* Striped to just extract the type
* Taken from: http://www.cet.nau.edu/~mc8/Socket/Tutorials/disect2.c
*
********************************************************************************/
u_int16_t handle_ethernet (u_char *args,const struct pcap_pkthdr* pkthdr,
                           const u_char* packet){
```

```
    u_int caplen = pkthdr->caplen;
    u_int length = pkthdr->len;
    struct ether_header *eptr;  /* net/ethernet.h */
    u_short ether_type;

    if (caplen < ETHER_HDRLEN)
    {
        snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Packet length less "\
                                         "than ethernet header length");
        call_sys_log(textbuffer);
        return -1;

    }//if (caplen < ETHER_HDRLEN)

    /* lets start with the ether header... */
    eptr = (struct ether_header *) packet;
    ether_type = ntohs(eptr->ether_type);

    if (TROUBLESHOOT==2){
       /* Lets print SOURCE DEST TYPE LENGTH */
       fprintf(stdout,"TROUBLESHOOT: ETH: ");
       fprintf(stdout,"%s ",ether_ntoa((struct ether_addr*)eptr->ether_shost));
       fprintf(stdout,"%s ",ether_ntoa((struct ether_addr*)eptr->ether_dhost));

       // check to see if we have an ip packet
       if (ether_type == ETHERTYPE_IP){

          fprintf(stdout,"(IP)");

       } else  if (ether_type == ETHERTYPE_ARP) {

         fprintf(stdout,"(ARP)");

       } else  if (eptr->ether_type == ETHERTYPE_REVARP) {

         fprintf(stdout,"(RARP)");

       } else {

         fprintf(stdout,"(?)");
      }

      fprintf(stdout," %d\n",length);

    } //if (TROUBLESHOOT==2){

    return ether_type;

}//u_int16_t handle_ethernet


/***************************************************************************
*
* Function to take apart the IP header!!
* Taken from: http://www.cet.nau.edu/~mc8/Socket/Tutorials/disect2.c
*
***************************************************************************/

u_char* handle_IP (u_char *args,const struct pcap_pkthdr* pkthdr,
                   const u_char* packet) {

    const struct my_ip* ip_header;
    u_int length = pkthdr->len;
    u_int hlen,off,version;
    int i;

    int len;
```

```
/* jump pass the ethernet header */
ip_header = (struct my_ip*)(packet + sizeof(struct ether_header));
length -= sizeof(struct ether_header);

//check to see we have a packet of valid length
if (length < sizeof(struct my_ip)) {

    snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: NOTE: *************"\
                                     "**truncated ip %d\n",length);
    call_sys_log(textbuffer);
    return NULL;

}//if (length < sizeof(struct my_ip)) {

len     = ntohs(ip_header->ip_len);
hlen    = IP_HL(ip_header); /* header length */
version = IP_V(ip_header);/* ip version */

if(version != 4) {//check version

  snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: We have an unknown"\
                                   " version %d\n",version);
  call_sys_log(textbuffer);
  return NULL;

}// if(version != 4) {

if(hlen < 5 ) { //check header length

    snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: We have a bad-hlen"\
                                     " %d \n",hlen);
    call_sys_log(textbuffer);

}//if(hlen < 5 ) {

if(length < len){//see if we have as much packet as we should

    snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: truncated IP - "\
                                     "%d bytes missing\n",len - length);
    call_sys_log(textbuffer);

}//if(length < len){

off = ntohs(ip_header->ip_off);
// Check to see if we have the first fragment aka no 1's in first 13 bits
if((off & 0x1fff) == 0 ){

   if (TROUBLESHOOT==3) {

      //print SOURCE DESTINATION hlen version len offset
      fprintf(stdout,"TROUBLESHOOT: IP: ");
      fprintf(stdout,"%s ", inet_ntoa(ip_header->ip_src));
      fprintf(stdout,"%s %d %d %d %d\n", inet_ntoa(ip_header->ip_dst), \
              hlen,version,len,off);

   }//    if (TROUBLESHOOT==3) {


   //Is the sender our destination!!!!
   if ((strcmp(tunnelip,inet_ntoa(ip_header->ip_src))==0) &&
               (TUNNEL_PROTO == ip_header->ip_p)){

       u_char* packet_to_pass = (u_char*) (packet +
                                sizeof(struct ether_header) + hlen * 4);
       handle_IP6(args,pkthdr,packet_to_pass);

   }//if ((strcmp(tunnelip,inet_ntoa(ip_header->ip_src))==0) && (TUNNEL_PRO
```

```
    }//    if((off & 0x1fff) == 0 ){ /* aka no 1's in first 13 bits */

    return NULL;

}//u_char* handle_IP (u_char *args,const struct pcap_pkthdr* pkthdr,const u_cha


u_char* handle_IP6 (u_char *args,const struct pcap_pkthdr* pkthdr,
                    const u_char* packet){

    struct ip6_hdr* ip6_header;

    ip6_header = (struct ip6_hdr*) packet;

    /*********************************************************************
    *
    * Grab contents of this packet
    *
    *********************************************************************/

    protocol    = ip6_header->ip6_ctlun.ip6_un1.ip6_un1_nxt;
    source      = ip6_header->ip6_src;
    destination = ip6_header->ip6_dst;

    if (TROUBLESHOOT==4) {

       char  temp_addr_name[INET6_ADDRSTRLEN];
       inet_ntop(AF_INET6, &destination , temp_addr_name, INET6_ADDRSTRLEN);
       fprintf(stderr,"TROUBLESHOOT: Received packet to ipv6 address: %s\n",\
                      temp_addr_name);
       inet_ntop(AF_INET6, &source , temp_addr_name, INET6_ADDRSTRLEN);
       fprintf(stderr,"TROUBLESHOOT: Received packet from ipv6 address: %s\n",\
                       temp_addr_name);
     fprintf(stderr,"TROUBLESHOOT: Received packet with nh as %i\n", protocol);


    }//if (TROUBLESHOOT==4) {

    if (cmpaddr(&destination, &our_ipv6_addr, 128) != ZERO){

       char  temp_addr_name[INET6_ADDRSTRLEN];
       inet_ntop(AF_INET6, &destination , temp_addr_name, INET6_ADDRSTRLEN);
       snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Received packet to "\
                "wrong ipv6 address: %s", temp_addr_name);
       call_sys_log(textbuffer);

    }  else {

       if (protocol == IPPROTO_ICMPV6) {

         if (TROUBLESHOOT==4){

            fprintf(stderr,"TROUBLESHOOT: Received ICMPv6 message\n");

         }//if (TROUBLESHOOT==4){

         u_char* packet_to_pass = (u_char*) (packet + sizeof(struct ip6_hdr));
         handle_ICMPv6(args,pkthdr,packet_to_pass);


       } else {

          char  temp_addr_name[INET6_ADDRSTRLEN];
          inet_ntop(AF_INET6, &source , temp_addr_name, INET6_ADDRSTRLEN);
          snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Received packet "\
                   "with unknown protocol type %i from address: %s",protocol,\
                    temp_addr_name);
          call_sys_log(textbuffer);
```

```
       }//if (protocol == IPPROTO_ICMPV6) {


    }//if (matchaddr(destination, our_ipv6_addr) != ZERO){

    return NULL;

}//u_char* handle_IP6 (u_char *args,const struct pcap_pkthdr* pkthdr,const u_ch


u_char* handle_ICMPv6 (u_char *args,const struct pcap_pkthdr* pkthdr,
                       const u_char* packet){

    struct icmp6_hdr* icmp6_header;

    icmp6_header = (struct icmp6_hdr*) packet;

    icmp_type = icmp6_header->icmp6_type;
    icmp_code = icmp6_header->icmp6_code;

    if (icmp_type == DESTINATION_UNREACHABLE ||
        icmp_type == TIME_EXCEEDED) {

       if (TROUBLESHOOT==5) {

          fprintf(stderr,"TROUBLESHOOT: Received packet with unreachable "\
                         "icmp_type %i and icmp_code %i\n", icmp_type, \
                         icmp_code);

       }//if (TROUBLESHOOT==5) {

       u_char* packet_to_pass = (u_char*) (packet + sizeof(struct icmp6_hdr));
       handle_ICMPv6_Unreachable(args,pkthdr,packet_to_pass);

    } else if (icmp_type == ECHO_REPLY) {

       icmp_ident    = ntohs(icmp6_header->icmp6_id);
       icmp_sequence = ntohs(icmp6_header->icmp6_seq);
       sendmessage();

       if (TROUBLESHOOT==5){

          fprintf(stderr,"TROUBLESHOOT: Received ICMPv6 Echo REPLY id = %i, "\
                         "seq=%i\n", icmp_ident, icmp_sequence);

       }//if (TROUBLESHOOT==5){

    } else {

       char  temp_addr_name[INET6_ADDRSTRLEN];
       inet_ntop(AF_INET6, &source , temp_addr_name, INET6_ADDRSTRLEN);
       snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Received ICMP with "\
                "unknown type %i from address: %s",icmp_type, temp_addr_name);
       call_sys_log(textbuffer);

       if (TROUBLESHOOT==5) {

          fprintf(stderr," They type matches:\n");
          if (icmp_type == 3) {

             fprintf(stderr,"Matched the 3\n");

          } else if (icmp_type == htons(3)) {

             fprintf(stderr, "Matched the htons 3\n");

          } else {

             fprintf(stderr, "Did not match anything XX%iXX\n", icmp_type);
```

```
          }//if (icmp_type == 3) {

       }//if (TROUBLESHOOT==5) {

    }//if (icmp_type == DESTINATION_UNREACHABLE) {
    return NULL;

}//u_char* handle_ICMPv6 (u_char *args,const struct pcap_pkthdr* pkthdr,const u


u_char* handle_ICMPv6_Unreachable (u_char *args,
                                   const struct pcap_pkthdr* pkthdr,
                                   const u_char* packet){


    struct ip6_hdr*   ip6_header;
    ip6_header   = (struct ip6_hdr*) packet;

    original_protocol    = ip6_header->ip6_ctlun.ip6_un1.ip6_un1_nxt;
    original_source      = ip6_header->ip6_src;
    original_destination = ip6_header->ip6_dst;
    original_hop_limit   = ip6_header->ip6_ctlun.ip6_un1.ip6_un1_hlim;

    if (TROUBLESHOOT==6) {

       char  temp_addr_name[INET6_ADDRSTRLEN];
       inet_ntop(AF_INET6, &original_destination , temp_addr_name,
                 INET6_ADDRSTRLEN);
       fprintf(stderr,"TROUBLESHOOT: Received original packet to "\
                      "ipv6 address: %s\n", temp_addr_name);
      inet_ntop(AF_INET6, &original_source , temp_addr_name, INET6_ADDRSTRLEN);
       fprintf(stderr,"TROUBLESHOOT: Received original packet from ipv6 "\
                      "address: %s\n", temp_addr_name);
       fprintf(stderr,"TROUBLESHOOT: Received original packet with nh as"\
                      " %i\n", original_protocol);


    }//if (TROUBLESHOOT==6) {

    if (original_protocol == IPPROTO_ICMPV6) {

       if (TROUBLESHOOT==6){

          fprintf(stderr,"TROUBLESHOOT: Received original ICMPv6 message\n");

       }//if (TROUBLESHOOT==6){
       u_char* packet_to_pass = (u_char*) (packet + sizeof(struct ip6_hdr));
       handle_ICMPv6_original(args,pkthdr,packet_to_pass);

    } else if (original_protocol == IPPROTO_UDP) {

       if (TROUBLESHOOT==6){

          fprintf(stderr,"TROUBLESHOOT: Received original UDP message\n");

       }//if (TROUBLESHOOT==6){

       u_char* packet_to_pass = (u_char*) (packet + sizeof(struct ip6_hdr));
       handle_UDP_original(args,pkthdr,packet_to_pass);

    } else if (original_protocol == LIBNET_IPV6_NH_ROUTING) {

       if (TROUBLESHOOT==6){

          fprintf(stderr,"TROUBLESHOOT: Received original routing message\n");

       }//if (TROUBLESHOOT==6){
       u_char* packet_to_pass = (u_char*) (packet + sizeof(struct ip6_hdr));
```

```
       handle_Routing_original(args,pkthdr,packet_to_pass);

    } else {

       char  temp_addr_name[INET6_ADDRSTRLEN];
       inet_ntop(AF_INET6, &source , temp_addr_name, INET6_ADDRSTRLEN);
       snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Received packet with"\
                " unknown original protocol type %i from address: %s",\
                original_protocol, temp_addr_name);
       call_sys_log(textbuffer);

    }//if (protocol == IPPROTO_ICMPV6) {

    return NULL;

}//u_char* handle_ICMPv6_Unreachable (u_char *args,const struct pcap_pkthdr* pk


u_char* handle_ICMPv6_original (u_char *args,const struct pcap_pkthdr* pkthdr,
                                const u_char* packet){

    struct icmp6_hdr* icmp6_header;

    icmp6_header = (struct icmp6_hdr*) packet;

    original_icmp_type = icmp6_header->icmp6_type;
    original_icmp_code = icmp6_header->icmp6_code;

    if (original_icmp_type == ECHO_REQUEST) {

       icmp_ident    = ntohs(icmp6_header->icmp6_id);
       icmp_sequence = ntohs(icmp6_header->icmp6_seq);

       sendmessage();

       if (TROUBLESHOOT==7){

          fprintf(stderr,"TROUBLESHOOT: Received original ICMPv6 Echo "\
                  "REQUEST type = %i, code = %i\n", original_icmp_type, \
                   original_icmp_code);
          fprintf(stderr,"TROUBLESHOOT: Received original ICMPv6 Echo "\
                  "REQUEST id = %i, seq=%i\n", icmp_ident, icmp_sequence);

       }//if (TROUBLESHOOT==7){

    } else {

       char  temp_addr_name[INET6_ADDRSTRLEN];
       inet_ntop(AF_INET6, &source , temp_addr_name, INET6_ADDRSTRLEN);
       snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Received original "\
                "ICMP with unknown type %i from address: %s",icmp_type, \
                temp_addr_name);
       call_sys_log(textbuffer);

    }//if (icmp_type == ECHO_REQUEST) {
    return NULL;

}//u_char* handle_ICMPv6_original (u_char *args,const struct pcap_pkthdr* pkthd


u_char* handle_UDP_original (u_char *args,const struct pcap_pkthdr* pkthdr,
                             const u_char* packet){

    struct udphdr*    udp_header;

    udp_header = (struct udphdr*) packet;

    original_source_port = ntohs(udp_header->source);
    original_dest_port   = ntohs(udp_header->dest);
```

```
    sendmessage();

    if (TROUBLESHOOT==8){

       fprintf(stderr,"TROUBLESHOOT: Received original UDP src port = %i, "\
              "dest port=%i\n", original_source_port, original_dest_port);

    }//if (TROUBLESHOOT==8){

    return NULL;

}//u_char* handle_UDP_original (u_char *args,const struct pcap_pkthdr* pkthdr,c


u_char* handle_Routing_original (u_char *args,const struct pcap_pkthdr* pkthdr,
                                 const u_char* packet){


    struct ip6_rthdr0*  route_header;
    route_header   = (struct ip6_rthdr0*) packet;

    unsigned int next_header             = route_header->ip6r0_nxt;
                 original_route_seg_left = route_header->ip6r0_segleft;
                 original_route_addr     = route_header->ip6r0_addr[0];

    if (TROUBLESHOOT==9) {

       char  temp_addr_name[INET6_ADDRSTRLEN+1];
       inet_ntop(AF_INET6, &original_route_addr , temp_addr_name,
                 INET6_ADDRSTRLEN);
       fprintf(stderr,"TROUBLESHOOT: Received original route packet with"\
                      " segments left %i\n", original_route_seg_left);
       fprintf(stderr,"TROUBLESHOOT: Received original route packet with "\
                      "nh as %i\n", next_header);
       fprintf(stderr,"TROUBLESHOOT: Received original route packet to ipv6"\
                      " address: %s\n", temp_addr_name);

    }//if (TROUBLESHOOT==9) {

    if (next_header == IPPROTO_ICMPV6) {

       if (TROUBLESHOOT==9){

          fprintf(stderr,"TROUBLESHOOT: Received original ICMPv6 message\n");

       }//if (TROUBLESHOOT==9){
       original_protocol = next_header;
       u_char* packet_to_pass = (u_char*) (packet + sizeof(struct ip6_rthdr0));
       handle_ICMPv6_original(args,pkthdr,packet_to_pass);

    } else if (next_header == IPPROTO_UDP) {

       if (TROUBLESHOOT==9){

          fprintf(stderr,"TROUBLESHOOT: Received original UDP message\n");

       }//if (TROUBLESHOOT==9){

       original_protocol = next_header;
       u_char* packet_to_pass = (u_char*) (packet + sizeof(struct ip6_rthdr0));
       handle_UDP_original(args,pkthdr,packet_to_pass);

    } else {

       char  temp_addr_name[INET6_ADDRSTRLEN];
       inet_ntop(AF_INET6, &source , temp_addr_name, INET6_ADDRSTRLEN);
       snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Received packet with"\
                " unknown original routing nh %i\n", next_header);
       call_sys_log(textbuffer);
```

```
    }//if (protocol == IPPROTO_ICMPV6) {

    return NULL;

}//u_char* handle_Routing_original (u_char *args,const struct pcap_pkthdr* pkth



void probe_receive() {


      /************************************************************************
      *
      * Define constants needed by the probe_receive
      *
      ************************************************************************/

      const BAD_STARTUP  = -1;   // Bad start we need to terminate
      const FOREVER      = -1;   // Tells PCAP_LOOP to receive until stop
      const GOOD_STARTUP =  0;   // Keep going everything good
      const u_char *packet;      // Our packet received by pcap

      /************************************************************************
      *
      * Define variables needed by probe_receive
      *
      ************************************************************************/

      int startup  = GOOD_STARTUP;

      /************************************************************************
      *
      * Define variables needed by the pcap
      *
      ************************************************************************/

      char *  device = OUTDEVICE;       // What device we will use for packets
      struct  bpf_program filterprogram;// Hold filter compiled program
      // This is much longer than will be needed
      char    filterstring[100]          = "";

      struct       pcap_pkthdr hdr;     // pcap header defined in pcap.h
      struct       ether_header *eptr;  // Ethernet header
      u_char       *ptr;                // printing out hardware header info
      //If we want to pass info to our callback then use this
      u_char*      args_for_callback         = NULL;
      // Buffer to hold error messages from PCAP
      char         caperrbuf[PCAP_ERRBUF_SIZE];
      bpf_u_int32  mask;                            // Our netmask
      bpf_u_int32  net;                               // Our IP address

      /************************************************************************
      *
      * Kick off the signal handler
      *
      ************************************************************************/

      signal(SIGKILL, receive_handler);
      signal(SIGQUIT, SIG_IGN);
     snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Starting probe receive");
      call_sys_log(textbuffer);


      /************************************************************************
      *
      * Get the Ipv6 address we started with
      *
```

```
*
**************************************************************************/

int check = inet_pton(AF_INET6, our_ipv6_addr_name, &our_ipv6_addr);

if (check <= ZERO) {

   snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: ERROR: Could not"\
                                    " convert our IPv6 addr");
   call_sys_log(textbuffer);
   startup = BAD_STARTUP;

}//if (check <= ZERO) {

/*************************************************************************
*
* Zero global values before we start
*
*
**************************************************************************/

source.in6_u.u6_addr32[0]                = ZERO;
source.in6_u.u6_addr32[1]                = ZERO;
source.in6_u.u6_addr32[2]                = ZERO;
source.in6_u.u6_addr32[3]                = ZERO;
destination.in6_u.u6_addr32[0]           = ZERO;
destination.in6_u.u6_addr32[1]           = ZERO;
destination.in6_u.u6_addr32[2]           = ZERO;
destination.in6_u.u6_addr32[3]           = ZERO;
protocol                                 = ZERO;
original_source.in6_u.u6_addr32[0]       = ZERO;
original_source.in6_u.u6_addr32[1]       = ZERO;
original_source.in6_u.u6_addr32[2]       = ZERO;
original_source.in6_u.u6_addr32[3]       = ZERO;
original_destination.in6_u.u6_addr32[0]  = ZERO;
original_destination.in6_u.u6_addr32[1]  = ZERO;
original_destination.in6_u.u6_addr32[2]  = ZERO;
original_destination.in6_u.u6_addr32[3]  = ZERO;
original_protocol                        = ZERO;
original_hop_limit                       = ZERO;
icmp_type                                = ZERO;
icmp_code                                = ZERO;
icmp_ident                               = ZERO;
icmp_sequence                            = ZERO;
original_source_port                     = ZERO;
original_dest_port                       = ZERO;
original_route_seg_left                  = ZERO;
original_route_addr.in6_u.u6_addr32[0]   = ZERO;
original_route_addr.in6_u.u6_addr32[1]   = ZERO;
original_route_addr.in6_u.u6_addr32[2]   = ZERO;
original_route_addr.in6_u.u6_addr32[3]   = ZERO;
original_icmp_type                       = ZERO;
original_icmp_code                       = ZERO;


/*************************************************************************
*
* Set up pcap and open up the session
*
*
**************************************************************************/

if (startup == GOOD_STARTUP){

   pcap_handle = pcap_open_live(device, PACKET_CAPTURE_SIZE,
                                NOT_PROMISCUOS, ZERO, caperrbuf);

   if (pcap_handle == NULL) {
```

```
      snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: We had an error "\
                                       "opening pcap: %s", caperrbuf);
      call_sys_log(textbuffer);
      startup = BAD_STARTUP;

   }//if (pcap_handle == NULL) {

}//if (startup == GOOD_STARTUP){

if (startup == GOOD_STARTUP){

   //Get our ip address
   if (pcap_lookupnet(device, &net, &mask, caperrbuf) == BAD_STARTUP) {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Couldn't get "\
                          "netmask for device %s: %s", device, caperrbuf);
      call_sys_log(textbuffer);
      startup = BAD_STARTUP;

   }//if (pcap_lookupnet(device, &net, &mask, caperrbuf) == BAD_STARTUP)

}//if (startup == GOOD_STARTUP){

/*************************************************************************
*
* Build the filter for the listener!!!
* Taken from:  http://www.cet.nau.edu/~mc8/Socket/Tutorials/section3.html
*
*************************************************************************/

if (startup == GOOD_STARTUP){

   //Build the filter string
   sprintf(filterstring,"src %s", tunnelip);

   //Build the filter
   if(pcap_compile(pcap_handle,&filterprogram,filterstring,ZERO,net) ==
      BAD_STARTUP){

      snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Error calling"\
                              " pcap_compile when building the filter");
      call_sys_log(textbuffer);
      startup = BAD_STARTUP;

   }//if(pcap_compile(pcap_handle,&filterprogram,filterstring,0,src_ip) =

}//if (startup == GOOD_STARTUP){

if (startup == GOOD_STARTUP){

   //Turn on the filter
   if(pcap_setfilter(pcap_handle,&filterprogram) == BAD_STARTUP){

      snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Error setting "\
                                       "filter for pcap!");
      call_sys_log(textbuffer);
      startup = BAD_STARTUP;

   }//if(pcap_setfilter(pcap_handle,&filterprogram) == -1){

}//if (startup == GOOD_STARTUP){


/*************************************************************************
*
* Grab the response!!!
* Taken from:  http://www.cet.nau.edu/~mc8/Socket/Tutorials/section3.html
*
*************************************************************************/
```

```
      if (startup == GOOD_STARTUP){

         pcap_loop(pcap_handle,FOREVER,my_callback,args_for_callback);

      }//if (startup == GOOD_STARTUP){

      if (startup == GOOD_STARTUP){

         snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Stopping probe "\
                                          "receive");

      } else {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Stopping probe "\
                                          "receive, bad startup");

      }//if (startup == GOOD_STARTUP){

      call_sys_log(textbuffer);


}//void probe_receive() {
```

PROBE_RECEIVE.H

```
/*****************************************************************************
 * This is the header file of the main program of the probing receiver
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_receive.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_PROBE_RECEIVE
#define INCLUSION_GUARD_PROGRAM_PROBE_RECEIVE

    void probe_receive();

#endif //INCLUSION_GUARD_PROGRAM_PROBE_RECEIVE
```

PROBE_RECEIVE_STUB.C

```
/*****************************************************************************
 * This is the packet generator program of the probing engine
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_receive.c
 *
 *****************************************************************************/

#include <string.h>
#include <netinet/ip6.h>
#include <netinet/udp.h>
#include <netinet/icmp6.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h> /* includes net/ethernet.h */
#include <netinet/ether.h>
#include <netinet/ip.h>
#include <string.h>
#include <netinet/tcp.h>
#include "../src/probe_receive.h"
#include "../src/probe_main.h"
#include "../src/probe_utils.h"
#include <pcap.h>

#define TROUBLESHOOT 0
#define TUNNEL_PROTO 41             // Type of packet for IPv6 tunnel
//This is the max size of the packet we want pcap to capture
#define PACKET_CAPTURE_SIZE 0xFFFF
#define NOT_PROMISCUOS 0            //If PCAP should be in promiscous mode

#ifndef ETHER_HDRLEN
#define ETHER_HDRLEN 14
#endif
#define MAX_BUF       65536 //Maximum buffer we can receive
 //  This needs to be 11 for alias tests and 2 for main tests
#define NUMBEROFPKTS 11


/*****************************************************************************
 *
 * Define global variables needed by the probe_receive
 *
 *****************************************************************************/

    // Buffer to hold our message foe syslog
    char                      textbuffer[MAX_MESSAGE];
    // The handle to our pcap session
    pcap_t                   *pcap_handle;
    // Our ipv6 address we started with
    struct in6_addr           our_ipv6_addr;
    // Source address of packet
    struct in6_addr           source;
    // Destination address of the packet
    struct in6_addr           destination;
    // The protocol carried by this packet
    unsigned int              protocol                = ZERO;
    // Original source address of packet
    struct in6_addr           original_source;
    // Original destination address of packet
    struct in6_addr           original_destination;
    // Original protocol carried by this packet
    unsigned int              original_protocol       = ZERO;
```

```
    // The remaining hop limit on packet
    unsigned int             original_hop_limit      = ZERO;
    // The icmp type of this message
    unsigned int             icmp_type               = ZERO;
    // The icmp code of this message
    unsigned int             icmp_code               = ZERO;
    // The id number of this icmp echo reply
    unsigned int             icmp_ident              = ZERO;
    // The sequence number of this icmp echo reply
    unsigned int             icmp_sequence           = ZERO;
    // The original source port of the sent packet
    unsigned int             original_source_port    = ZERO;
    // The original dest port of packet
    unsigned int             original_dest_port      = ZERO;
    // The # of segements left in original packet
    unsigned int             original_route_seg_left = ZERO;
    // The next sgement address in route header
    struct in6_addr          original_route_addr;
    // The value of the original icmp type
    unsigned int             original_icmp_type      = ZERO;
    // The value of the original icmp code
    unsigned int             original_icmp_code      = ZERO;

             int             choice                  = ZERO;
             int             done                    = FALSE;
             int             check                   = ZERO;
             char            tempip[INET6_ADDRSTRLEN + 1];

/*****************************************************************************
 *
 * This is so if main tells us to stop we handle it and die gracefully
 *
 *****************************************************************************/
void message_handler(int sig) {

   /*************************************************************************
    *
    * If we figure out how to interupt pcap_loop then change the intrupt, for
    * now this serves no function
    *
    *************************************************************************/


    int  status;
    wait(&status);
    pcap_breakloop(pcap_handle);

}//void message_handler(int sig) {


/*****************************************************************************
 *
 * Function to send messages back to the main program
 *
 *****************************************************************************/

void sendmessage(void) {


   int                      sendsuccess      = ZERO;
   //Buffer for messages to the main function
   struct receive_buffer    *message_to_main;
   struct messagebuffer     sendbuffer;


   message_to_main = (struct receive_buffer*) (sendbuffer.text);
   sendbuffer.type = RECEIVE;

   /*************************************************************************
```

```
*
* Prep message and zero our global values for next receive
*
*****************************************************************************/

message_to_main->source                 = source;
source.in6_u.u6_addr32[0]               = ZERO;
source.in6_u.u6_addr32[1]               = ZERO;
source.in6_u.u6_addr32[2]               = ZERO;
source.in6_u.u6_addr32[3]               = ZERO;
message_to_main->destination            = destination;
destination.in6_u.u6_addr32[0]          = ZERO;
destination.in6_u.u6_addr32[1]          = ZERO;
destination.in6_u.u6_addr32[2]          = ZERO;
destination.in6_u.u6_addr32[3]          = ZERO;
message_to_main->protocol               = protocol;
protocol                                = ZERO;
message_to_main->original_source        = original_source;
original_source.in6_u.u6_addr32[0]      = ZERO;
original_source.in6_u.u6_addr32[1]      = ZERO;
original_source.in6_u.u6_addr32[2]      = ZERO;
original_source.in6_u.u6_addr32[3]      = ZERO;
message_to_main->original_destination   = original_destination;
original_destination.in6_u.u6_addr32[0] = ZERO;
original_destination.in6_u.u6_addr32[1] = ZERO;
original_destination.in6_u.u6_addr32[2] = ZERO;
original_destination.in6_u.u6_addr32[3] = ZERO;
message_to_main->original_protocol      = original_protocol;
original_protocol                       = ZERO;
message_to_main->original_hop_limit     = original_hop_limit;
original_hop_limit                      = ZERO;
message_to_main->icmp_type              = icmp_type;
icmp_type                               = ZERO;
message_to_main->icmp_code              = icmp_code;
icmp_code                               = ZERO;
message_to_main->icmp_ident             = icmp_ident;
icmp_ident                              = ZERO;
message_to_main->icmp_sequence          = icmp_sequence;
icmp_sequence                           = ZERO;
message_to_main->original_source_port   = original_source_port;
original_source_port                    = ZERO;
message_to_main->original_dest_port     = original_dest_port;
original_dest_port                      = ZERO;
message_to_main->original_route_seg_left = original_route_seg_left;
original_route_seg_left                 = ZERO;
message_to_main->original_route_addr    = original_route_addr;
original_route_addr.in6_u.u6_addr32[0]  = ZERO;
original_route_addr.in6_u.u6_addr32[1]  = ZERO;
original_route_addr.in6_u.u6_addr32[2]  = ZERO;
original_route_addr.in6_u.u6_addr32[3]  = ZERO;
message_to_main->original_icmp_type     = original_icmp_type;
original_icmp_type                      = ZERO;
message_to_main->original_icmp_code     = original_icmp_code;
original_icmp_code                      = ZERO;


/*****************************************************************************
*
* Send the message
*
*****************************************************************************/

sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                     sizeof(struct receive_buffer), IPC_NOWAIT);

if(sendsuccess != ZERO){

   snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: ERROR: Message send"\
                                    " to main failed.  Error: ");
```

```
      fprintf(stderr,"%s\n", textbuffer);

   } else {

      snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Message was sent to"\
                                       " main");
      fprintf(stderr,"%s\n", textbuffer);

   }//if(sendsuccess != ZERO){


}//void sendmessage() {


void probe_receive() {


      /*********************************************************************
      *
      * Kick off the signal handler
      *
      *********************************************************************/

      signal(SIGKILL, message_handler);
      signal(SIGQUIT, SIG_IGN);
      // Needed to make sure the probe generator sends the probes first
      sleep(3);
     snprintf(textbuffer, MAX_MESSAGE,"PROBE RECEIVE: Starting probe receive");
      call_sys_log(textbuffer);


      /*********************************************************************
      *
      * Set Values!
      *
      *
      *********************************************************************/
      int my_count = -1;

      while (done == FALSE) {

         if (my_count >= NUMBEROFPKTS) {

            fprintf(stderr,"I am going to sleep\n");
            // Needed to make sure the probe generator sends the probes first
            sleep(3);
            my_count = ZERO;

         } else {

            my_count++;
            fprintf(stderr,"I just increment my_count\n");

         }//if (my_count >= 3) {
         fprintf(stderr,"My count = %i\n", my_count);
         printf("Do you send to main or quit?\n");
         printf("SEND to MAIN      =  1\n");
         printf("QUIT              =  2\n");
         printf("Entry             =  ");
         scanf ("%d", &choice);
         printf("\n");
         if (choice == 2) {

            done = TRUE;

         }//if (choice == 2) {

         if (done == FALSE) {
```

```
check = ZERO;
while (check <= ZERO) {

   printf("Source IPv6 address you want to pass: ");
   scanf("%46s", tempip);
   printf("\n");
   check = inet_pton(AF_INET6, tempip, &source);
   if (check <= ZERO){

      fprintf(stderr,"Error in the address passed in, do again\n");
      fprintf(stderr,"Input string = %s\n", tempip);

   }//if (check != ZERO){
}// while (check != ZERO) {
check = ZERO;
while (check <= ZERO) {

   printf("Destination IPv6 address you want to pass: ");
   scanf("%46s", tempip);
   printf("\n");
   check = inet_pton(AF_INET6, tempip, &destination);
   if (check <= ZERO){

      fprintf(stderr,"Error in the address passed in, do again\n");
      fprintf(stderr,"Input string = %s\n", tempip);

   }//if (check != ZERO){
}// while (check != ZERO) {
printf("What protocol do you want?\n");
scanf ("%d", &choice);
printf("\n");
protocol = choice;
check = ZERO;
while (check <= ZERO) {

   printf("Original Source IPv6 address you want to pass: ");
   scanf("%46s", tempip);
   printf("\n");
   check = inet_pton(AF_INET6, tempip, &original_source);
   if (check <= ZERO){

      fprintf(stderr,"Error in the address passed in, do again\n");
      fprintf(stderr,"Input string = %s\n", tempip);

   }//if (check != ZERO){
}// while (check != ZERO) {
check = ZERO;
while (check <= ZERO) {

   printf("Original Destination IPv6 address you want to pass: ");
   scanf("%46s", tempip);
   printf("\n");
   check = inet_pton(AF_INET6, tempip, &original_destination);
   if (check <= ZERO){

      fprintf(stderr,"Error in the address passed in, do again\n");
      fprintf(stderr,"Input string = %s\n", tempip);

   }//if (check != ZERO){
}// while (check != ZERO) {
printf("What original protocol do you want?\n");
scanf ("%d", &choice);
printf("\n");
original_protocol = choice;
printf("What original hop limit do you want?\n");
scanf ("%d", &choice);
printf("\n");
original_hop_limit = choice;
```

```
printf("What icmp type do you want?\n");
scanf ("%d", &choice);
printf("\n");
icmp_type = choice;
printf("What icmp code do you want?\n");
scanf ("%d", &choice);
printf("\n");
icmp_code = choice;
printf("What icmp ident do you want?\n");
scanf ("%d", &choice);
printf("\n");
icmp_ident = choice;
printf("What icmp sequence do you want?\n");
scanf ("%d", &choice);
printf("\n");
icmp_sequence = choice;
printf("What original source port do you want?\n");
scanf ("%d", &choice);
printf("\n");
original_source_port = choice;
printf("What original destination port do you want?\n");
scanf ("%d", &choice);
printf("\n");
original_dest_port = choice;
printf("What original route segements left do you want?\n");
scanf ("%d", &choice);
printf("\n");
original_route_seg_left = choice;
check = ZERO;
while (check <= ZERO) {

   printf("Original route IPv6 address you want to pass: ");
   scanf("%46s", tempip);
   printf("\n");
   check = inet_pton(AF_INET6, tempip, &original_route_addr);
   if (check <= ZERO){

      fprintf(stderr,"Error in the address passed in, do again\n");
      fprintf(stderr,"Input string = %s\n", tempip);

   }//if (check != ZERO){
}// while (check != ZERO) {
printf("What original icmp type do you want?\n");
scanf ("%d", &choice);
printf("\n");
original_icmp_type = choice;
printf("What original icmp code do you want?\n");
scanf ("%d", &choice);
printf("\n");
original_icmp_code = choice;

printf("Sending the following message:\n");
char  temp_addr_name[INET6_ADDRSTRLEN+1];
inet_ntop(AF_INET6, &source , temp_addr_name, INET6_ADDRSTRLEN);
printf("source address = %s\n", temp_addr_name);
inet_ntop(AF_INET6, &destination , temp_addr_name,
          INET6_ADDRSTRLEN);
printf("destination address = %s\n", temp_addr_name);
printf("protocol = %i\n", protocol);
inet_ntop(AF_INET6, &original_source , temp_addr_name,
          INET6_ADDRSTRLEN);
printf("original source address = %s\n", temp_addr_name);
inet_ntop(AF_INET6, &original_destination , temp_addr_name,
          INET6_ADDRSTRLEN);
printf("original destination address = %s\n", temp_addr_name);
printf("original protocol = %i\n", original_protocol);
printf("original hop limit = %i\n", original_hop_limit);
printf("icmp type = %i\n", icmp_type);
printf("icmp code = %i\n", icmp_code);
```

```
            printf("////////////////////////////////////////////////////////\
                   "//////////////////\n");
            printf("icmp ident = %i\n", icmp_ident);
            printf("icmp sequence = %i\n", icmp_sequence);
            printf("original source port = %i\n", original_source_port);
            printf("original dest port = %i\n", original_dest_port);
            printf("original route seg left = %i\n", original_route_seg_left);
            inet_ntop(AF_INET6, &original_route_addr , temp_addr_name,
                      INET6_ADDRSTRLEN);
            printf("original route address = %s\n", temp_addr_name);
            printf("original icmp type = %i\n", original_icmp_type);
            printf("original icmp code = %i\n", original_icmp_code);
            sendmessage();

      }//if (done == FALSE) {

   }//      while (done == FALSE) {


}//void probe_receive() {
```

PROBE_STOP.C

```
/*****************************************************************************
 * This is the module that decides if we should stop probing
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_stop.c
 *
 *****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <sys/msg.h>
#include <errno.h>
#include <netinet/in.h>
#include "probe_main.h"
#include "probe_stop.h"
#include "probe_utils.h"
#include "sys_log.h"


#define TROUBLESHOOT 0                  // 1 = global_stop
                                        // 2 = local_stop


/*****************************************************************************
 *
 * Define externals and globals
 *
 *****************************************************************************/

   // Set by calls to the library functions to define errors encountered
   extern int errno;

   // Buffer to hold our message for syslog
          char              textbuffer[MAX_MESSAGE];

/*****************************************************************************
 *
 * Subroutines
 *
 *****************************************************************************/

   int global_stop(const struct in6_addr address,
                   const struct in6_addr dest_address ){


     // Index into the global array to locate the item
     int             middle        = ZERO;
     // The result of a comparison
     int             result        = ZERO;
     int             count         = ZERO;  // Counter for loops
     // Address we will use for comparison
     struct in6_addr temp_addr;

     if (TROUBLESHOOT==1) {

        char *out_string             = NULL;
              out_string             = (char *) malloc (MAX_LINE_LENGTH);
        fprintf(stderr, "We are starting global_stop\n");
        if (inet_ntop(AF_INET6, &address , out_string, INET6_ADDRSTRLEN) ==
            NULL){

           fprintf(stderr," The first address passed in was not valid!\n");

        } else {
```

```
      fprintf (stderr, "The first address passed in was : %s\n", \
                       out_string);

   }//if (inet_ntop(AF_INET6, &address , out_string, INET6_ADDRSTRLEN) ==

   if (inet_ntop(AF_INET6, &dest_address , out_string, INET6_ADDRSTRLEN)
       == NULL){

      fprintf(stderr," The destination address passed in was not "\
                     "valid!\n");

   } else {

      fprintf (stderr, "The destination address passed in was : %s\n", \
                       out_string);

   }//if (inet_ntop(AF_INET6, &dest_address , out_string, INET6_ADDRSTRLEN
   free(out_string);


}//if (TROUBLESHOOT==1) {

if (number_of_global_stops == ZERO) {

   /*********************************************************************
   *
   * Deal with start up
   *
   *********************************************************************/

   global_stop_list[number_of_global_stops].address = address;
   global_stop_list[number_of_global_stops].dest_address = dest_address;
   number_of_global_stops++;
   return NOT_FOUND_AND_INSERTED;

} else {

   /*********************************************************************
   *
   * Locate the middle of the globals
   *
   *********************************************************************/

   int bottom = ZERO;
   int top    = (number_of_global_stops - 1);

   while (bottom <= top) {

      middle = (bottom + top) / 2;
     result = cmpaddr(&address, &(global_stop_list[middle].address), 128);
      if (TROUBLESHOOT==1) {

        fprintf(stderr, "The value of bottom = %i\n", bottom);
        fprintf(stderr, "The value of top    = %i\n", top);
        fprintf(stderr, "The value of middle = %i\n", middle);
        fprintf(stderr, "The value of result = %i\n", result);

      }//if (TROUBLESHOOT==1) {
      if (result == ZERO){

         /*****************************************************************
         *
         * If we are here we found the node but need to see if the
         * destination is in the list.  First locate the limits of
         * all the nodes with this address on the list
         *
         *****************************************************************/
```

```
bottom = middle;
if (bottom > ZERO) {

   while ((result == ZERO) && (bottom > ZERO)) {

      bottom--;
      result = cmpaddr(&address,
                       &(global_stop_list[bottom].address), 128);
      if (TROUBLESHOOT==1) {

         char *out_string             = NULL;
               out_string             =
                                (char *) malloc (MAX_LINE_LENGTH);

         fprintf(stderr, "We are are looking for bottom after"\
                         " match\n");
         if (inet_ntop(AF_INET6, &address,
                       out_string, INET6_ADDRSTRLEN) == NULL){

            fprintf(stderr," The address passed into cmp was "\
                           "not valid!\n");

         } else {

            fprintf (stderr, "The address passed into cmp was "\
                             ": %s\n", out_string);

         }//if (inet_ntop(AF_INET6, &address , out_string, INET6_

         if (inet_ntop(AF_INET6,
                       &(global_stop_list[bottom].address),
                       out_string, INET6_ADDRSTRLEN) == NULL){

            fprintf(stderr," The second address passed into cmp"\
                           " was not valid!\n");

         } else {

            fprintf (stderr, "The second address passed into "\
                             "cmp was : %s\n", out_string);

         }//if (inet_ntop(AF_INET6, &dest_address , out_string, I
         free(out_string);
         fprintf(stderr, "The result returned by cmp was: "\
                         "%i\n", result);

      }//if (TROUBLESHOOT==1) {

   }//while (result == ZERO) {
   if (result != ZERO) {

      bottom++; // The bottom of nodes with this address is here

   }//if (result != ZERO) {

}//if (bottom > ZERO) {
if (TROUBLESHOOT==1) {

   char *out_string             = NULL;
         out_string             =
                                (char *) malloc (MAX_LINE_LENGTH);

   fprintf(stderr, "The bottom value was\n");
   if (inet_ntop(AF_INET6, &(global_stop_list[bottom].address),
                 out_string, INET6_ADDRSTRLEN) == NULL){

      fprintf(stderr," The address for bottom is not valid!\n");

   } else {
```

```
      fprintf (stderr, "The address for bottom is : %s\n", \
                       out_string);

   }//if (inet_ntop(AF_INET6, &dest_address , out_string, INET6_A
   free(out_string);
   fprintf(stderr, "The value of bottom is: %i\n", bottom);

}//if (TROUBLESHOOT==1) {

top     = middle;
result  = ZERO;
if (top < (number_of_global_stops - 1)) {

   while ((result == ZERO) &&
           (top < (number_of_global_stops - 1))) {

      top++;
      result = cmpaddr(&address,
                       &(global_stop_list[top].address), 128);
      if (TROUBLESHOOT==1) {

         char *out_string             = NULL;
               out_string             =
                               (char *) malloc (MAX_LINE_LENGTH);

         fprintf(stderr, "We are are looking for top after "\
                         "match\n");
         if (inet_ntop(AF_INET6, &address,
                       out_string, INET6_ADDRSTRLEN) == NULL){

            fprintf(stderr," The address passed into cmp was"\
                           " not valid!\n");

         } else {

            fprintf (stderr, "The address passed into cmp was :"\
                             " %s\n", out_string);

         }//if (inet_ntop(AF_INET6, &address , out_string, INET6_

         if (inet_ntop(AF_INET6,
                       &(global_stop_list[top].address),
                       out_string, INET6_ADDRSTRLEN) == NULL){

            fprintf(stderr," The second address passed into "\
                           "cmp was not valid!\n");

         } else {

            fprintf (stderr, "The second address passed into "\
                             "cmp was : %s\n", out_string);

         }//if (inet_ntop(AF_INET6, &dest_address , out_string, I
         free(out_string);
         fprintf(stderr, "The result returned by cmp was: %i\n",\
                          result);

      }//if (TROUBLESHOOT==1) {

   }//while (result == ZERO) {
   if (result != ZERO) {

      top--; // The top of nodes with this address is here

   }//if (result != ZERO) {

}//if (top < (number_of_global_stops - 1)) {
```

```
if (TROUBLESHOOT==1) {

   char *out_string             = NULL;
         out_string             =
                               (char *) malloc (MAX_LINE_LENGTH);

   fprintf(stderr, "The top value was\n");
   if (inet_ntop(AF_INET6, &(global_stop_list[bottom].address),
                 out_string, INET6_ADDRSTRLEN) == NULL){

      fprintf(stderr," The address for top is not valid!\n");

   } else {

      fprintf (stderr, "The address for top is : %s\n", \
                       out_string);

   }//if (inet_ntop(AF_INET6, &dest_address , out_string, INET6_A
   free(out_string);
   fprintf(stderr, "The value of top is: %i\n", top);

}//if (TROUBLESHOOT==1) {

int found_it = FALSE;
for(count=bottom;count<=top;count++) {

   result = cmpaddr(&dest_address,
            &(global_stop_list[count].dest_address), 128);
   if (TROUBLESHOOT==1) {

      char *out_string             = NULL;
            out_string             =
                                  (char *) malloc (MAX_LINE_LENGTH);

      fprintf(stderr, "We are now in the loop looking for dest"\
                      " address match\n");
      if (inet_ntop(AF_INET6, &dest_address , out_string,
                    INET6_ADDRSTRLEN) == NULL){

         fprintf(stderr," The address passed into cmp was not "\
                        "valid!\n");

      } else {

         fprintf (stderr, "The address passed into cmp"\
                          " was : %s\n", out_string);

      }//if (inet_ntop(AF_INET6, &address , out_string, INET6_ADD

      if (inet_ntop(AF_INET6,
                    &(global_stop_list[count].dest_address),
                    out_string, INET6_ADDRSTRLEN) == NULL){

         fprintf(stderr," The second address passed into cmp "\
                        "was not valid!\n");

      } else {

         fprintf (stderr, "The second address passed into "\
                          "cmp was : %s\n", out_string);

      }//if (inet_ntop(AF_INET6, &dest_address , out_string, INET
      free(out_string);
      fprintf(stderr, "The result returned by cmp was: %i\n",\
                       result);

   }//if (TROUBLESHOOT==1) {
   if (result == ZERO) {
```

```
            return FOUND;

         } else if (result > ZERO){

            middle = count;
            found_it = TRUE;

         }//if (result == ZERO) {

      }//for(count=bottom;count<=top;count++) {
      if ((found_it = FALSE) && (bottom != top)) {

         middle = top + 1;

      } else if (result < ZERO) {

         middle = bottom;

      } else {

         middle = bottom + 1;

      }//if ((found_it = FALSE) && (bottom != top)) {
      if (TROUBLESHOOT==1) {

         printf("The value of middle is %i\n", middle);
         printf("The value of number_of_global_stops = %i\n", \
                 number_of_global_stops);

      }//if (TROUBLESHOOT==1) {
      for(count=number_of_global_stops;count>middle;count--) {

         global_stop_list[count].address      =
                                  global_stop_list[(count-1)].address;
         global_stop_list[count].dest_address =
                              global_stop_list[(count-1)].dest_address;

      }//for(count=number_of_global_stops;count>bottom;count--) {
      global_stop_list[middle].address      = address;
      global_stop_list[middle].dest_address = dest_address;
      number_of_global_stops++;
      return NOT_FOUND_AND_INSERTED;

   } else if (result < ZERO) {

      top = middle - 1;

   } else {

      bottom = middle + 1;

   }//if (result == ZERO){

}//while (bottom <= top) {

/*********************************************************************
*
* If we are here we did not find the node and need to insert it
*
*********************************************************************/

if (bottom > number_of_global_stops) {

   global_stop_list[number_of_global_stops].address      = address;
  global_stop_list[number_of_global_stops].dest_address = dest_address;
   number_of_global_stops++;
   return NOT_FOUND_AND_INSERTED;

} else {
```

```
         for(count=number_of_global_stops;count>bottom;count--) {

            global_stop_list[count].address      =
                                       global_stop_list[(count-1)].address;
            global_stop_list[count].dest_address =
                                  global_stop_list[(count-1)].dest_address;

         }//for(count=number_of_global_stops;count>bottom;count--) {
         global_stop_list[bottom].address      = address;
         global_stop_list[bottom].dest_address = dest_address;
         number_of_global_stops++;
         return NOT_FOUND_AND_INSERTED;

      }//if (bottom > number_of_global_stops) {

   }//if (number_of_global_stops == ZERO) {


   if (TROUBLESHOOT==1) {

      fprintf(stderr, "We are ending global_stop\n");

   }//if (TROUBLESHOOT==1) {


}//int global_stop(struct in6_addr address, struct in6_addr dest_address ){


int local_stop(const struct in6_addr in_address, const int source){

   struct stops      *temp_stop_pointer;
   struct stops      *last_stop_pointer;
   int               result;

   if (source_list[source].stop_next == NULL) {

      temp_stop_pointer = (struct stops*) malloc(sizeof(struct stops));

      if (temp_stop_pointer == NULL) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE STOP: ERROR: Unable to "\
                  "allocate memory to add local stop set for source %i", \
                  source);
         call_sys_log(textbuffer);
         return -1;

      } else {

         source_list[source].stop_next = temp_stop_pointer;

      }//if (last_stop_pointer->next == NULL) {

   } else {

      temp_stop_pointer = source_list[source].stop_next;
      last_stop_pointer = source_list[source].stop_next;
      while (temp_stop_pointer != NULL) {

         result = cmpaddr(&in_address, &(temp_stop_pointer->address), 128);
         if (result == ZERO){

            return FOUND;

         } else {

            last_stop_pointer = temp_stop_pointer;
            temp_stop_pointer = temp_stop_pointer->next;
```

```
         }//if (result == ZERO){

      }//while (temp_stop_pointer != NULL) {

      last_stop_pointer->next = (struct stops*) malloc(sizeof(struct stops));

      if (last_stop_pointer->next == NULL) {

         snprintf(textbuffer, MAX_MESSAGE,"PROBE STOP: ERROR: Unable to "\
                  "allocate memory to add local stop set for source %i", \
                  source);
         call_sys_log(textbuffer);
         return -1;

      } else {

         temp_stop_pointer = last_stop_pointer->next;

      }//if (last_stop_pointer->next == NULL) {

   }//if (source_list[source].stop_next == NULL) {

   temp_stop_pointer->next    = NULL;
   temp_stop_pointer->address = in_address;
   return NOT_FOUND_AND_INSERTED;

}//int local_stop(const struct in6_addr address, const int source){
```

PROBE_STOP.H

```
/*****************************************************************************
 * This is the header file for the program that decides if we should stop
 * probing
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_stop.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_PROBE_STOP
#define INCLUSION_GUARD_PROGRAM_PROBE_STOP

   int global_stop(const struct in6_addr address,
                   const struct in6_addr dest_address );
   int local_stop(const struct in6_addr in_address, const int source);

#endif //INCLUSION_GUARD_PROGRAM_PROBE_STOP
```

PROBE_UTILS.C

```
/*****************************************************************************
 * This contains all the utilities needed by the routines
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_utils.c
 *
 *****************************************************************************/

#include <sys/ipc.h>
#include <stdlib.h>
#include <stdio.h>
#include "probe_main.h"
#include "probe_utils.h"

#define TROUBLESHOOT 0              // 1 = call_sys_log
                                    // 2 = calc_p_value
                                    // 3 = print_IPC_error

/*****************************************************************************
 *
 * Used for qsort to define how to compare the two values
 *
 *****************************************************************************/

int intcmp(const void *v1, const void *v2) {

     return (*(int *)v1 - *(int *)v2);

}//int intcmp(const void *v1, const void *v2) {

/*****************************************************************************
 *
 *  Note: Code below modified from original java source code
 *  From:  https://www-rp.lip6.fr/site_npa/site_rp/tracerouteathome-1.0.tar.gz.
 *  Original code name was: computeHValue
 *  Copyright below:
 *
 * Copyright (c) 2005, Universit Pierre & Marie Curie - LiP6/CNRS
 * All rights reserved.
 * Redistribution and use in source and binary forms, with or
 * without modification, are permitted provided that the following
 * conditions are met:
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of the Universit Pierre & Marie Curie - LiP6/CNRS
 *    nor the names of its contributors may be used to endorse or
 *    promote products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```
 * POSSIBILITY OF SUCH DAMAGE.
 ****************************************************************************/

int calc_p_value(const int hop_length[], const int number_of_values) {

   int     count                         = ZERO; // General counter for looping
   int     temp_hop[MAX_HOPS];                   // Temporary array we will sort
   double  sum                           = 0.0;
   double  value                         = 0.0;

   if (TROUBLESHOOT==2) {

      for(count=ZERO;count<MAX_HOPS;count++) {

         printf("The unsorted value of %i element is %i\n", \
                (count+1), hop_length[count]);

      }//for(count=ZERO;count<MAX_HOPS;count++) {

   }//if (TROUBLESHOOT==2) {

   for(count=ZERO;count<MAX_HOPS;count++){

      temp_hop[count] = hop_length[count];

   }//for(count=ZERO;count<MAX_HOPS;count++)

   qsort(temp_hop, MAX_HOPS, sizeof(int), intcmp);

   if (TROUBLESHOOT==2) {

      for(count=ZERO;count<MAX_HOPS;count++) {

         printf("The sorted value of %i element is %i\n", \
                                (count+1), temp_hop[count]);

      }//for(count=ZERO;count<MAX_HOPS;count++) {

   }//if (TROUBLESHOOT==2) {

   for(count=ZERO; count< MAX_HOPS;count++) {

      value = temp_hop[count];
      sum  += value / number_of_values;

      if (TROUBLESHOOT==2) {

         fprintf(stderr,"The value of sum is %f\n", sum);

      }//if (TROUBLESHOOT==2) {

      if (sum >= DEFAULT_P_VALUE) {

         int local_count = ZERO;
         for(local_count=ZERO; local_count<MAX_HOPS; local_count++){

            if(hop_length[local_count] == temp_hop[count]){

               return local_count;

            }//if(hop_length[local_count] == temp_hop[count]){

         }//for(local_count=ZERO; local_count<MAX_HOPS, local_count++){
         return -1;

      }//if (sum >= DEFAULT_P_VALUE) {

   }//for(count=ZERO; count< MAX_HOPS;count++) {
```

```
}//int calc_p_value(int *hop_length[], int number_of_values) {

/*****************************************************************************
 *
 * End of copyrighted code
 *
 *****************************************************************************/

void print_IPC_error(char* program_name){

         char   textbuffer[MAX_MESSAGE];  // Buffer to hold our message

         if (errno == E2BIG) {

            snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a E2BIG "\
                     "error from IPC", program_name);
            call_sys_log(textbuffer);

            if (TROUBLESHOOT==3) {

               fprintf(stderr,"%s: The value of mtext is greater than msgsz"\
                       " and (msgflg & MSG_NOERâ€ ROR) is 0.\n", program_name);

            }//if (TROUBLESHOOT==3) {

         } else if (errno == EACCES) {

            snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a EACCES"\
                     " error from IPC", program_name);
            call_sys_log(textbuffer);

            if (TROUBLESHOOT==3) {

               fprintf(stderr,"%s: A  message  queue  identifier  exists  for"\
                       " the argument key, but\n", program_name);
               fprintf(stderr,"operation permission as specified by the  "\
                       "low-order  9  bits  of\n");
               fprintf(stderr,"message flag  would not be granted\n");

            }////if (TROUBLESHOOT==3) {

         } else if (errno == EAGAIN) {

            snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a EAGAIN"\
                     " error from IPC", program_name);
            call_sys_log(textbuffer);

            if (TROUBLESHOOT==3) {

               fprintf(stderr,"%s: The message cannot be sent for one of "\
                              "the reasons cited above\n", program_name);
               fprintf(stderr,"and (msgflg & IPC_NOWAIT) is non-zero.\n");

            }//if (TROUBLESHOOT==3) {

         } else if (errno == EEXIST) {

            snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a EEXIST "\
                     "error from IPC", program_name);
            call_sys_log(textbuffer);

            if (TROUBLESHOOT==3) {

               fprintf(stderr,"%s: A message queue identifier  exists  for"\
                              "  the  argument  key  but\n", program_name);
               fprintf(stderr,"((msgflg & IPC_CREAT) && (msgflg & IPC_EXCL))"\
                              " is non-zero\n");

            }//if (TROUBLESHOOT==3) {
```

```
} else if (errno == EIDRM) {

   snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a EIDRM "\
            "error from IPC", program_name);
   call_sys_log(textbuffer);

   if (TROUBLESHOOT==3) {

      fprintf(stderr,"%s: The message queue identifier msqid is "\
                     "removed from the system.\n", program_name);

   }//if (TROUBLESHOOT==3) {

} else if (errno == EINTR) {

   snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a EINTR "\
            "error from IPC", program_name);
   call_sys_log(textbuffer);

   if (TROUBLESHOOT==3) {

      fprintf(stderr,"%s: The msgsnd() or nsgrcv function was "\
              "interrupted by a signal.\n", program_name);

   }//if (TROUBLESHOOT==3) {

} else if (errno == EINVAL) {

   snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a EINVAL"\
            " error", program_name);
   call_sys_log(textbuffer);

   if (TROUBLESHOOT==3) {

      fprintf(stderr,"%s: The  value  of msqid is not a valid "\
                     "message queue identifier,\n", program_name);
      fprintf(stderr,"or the value of mtype is less than 1; or the "\
                     "value of msgsz is less\n");
      fprintf(stderr,"than 0 or greater than the system-imposed "\
                     "limit.\n");
      fprintf(stderr,"%s: An invalid signal was specified if from"\
                     " signal handler\n", program_name);

   }//if (TROUBLESHOOT==3) {

} else if (errno == ENOMSG) {

   snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a ENOMSG "\
                     "error from IPC", program_name);
   call_sys_log(textbuffer);

   if (TROUBLESHOOT==3) {

      fprintf(stderr,"%s: The  queue  does  not  contain a message "\
                     "of the desired type and\n", program_name);
      fprintf(stderr,"(msgflg & IPC_NOWAIT) is non-zero.\n");

   }//if (TROUBLESHOOT==3) {

} else if (errno == ENOENT) {

   snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a ENOENT "\
                                     "error from IPC", program_name);
   call_sys_log(textbuffer);

   if (TROUBLESHOOT==3) {

      fprintf(stderr,"%s: A  message  queue identifier does not "\
```

```
                        "exist for the argument key\n", program_name);
            fprintf(stderr,"and (msgflg & IPC_CREAT) is 0\n");

         }//if (TROUBLESHOOT==3) {

      } else if (errno == ENOSPC) {

         snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a ENOSPC"\
                                          " error from IPC", program_name);
         call_sys_log(textbuffer);

         if (TROUBLESHOOT==3) {

            fprintf(stderr,"%s: A message queue identifier is to  be  "\
                        "created  but  the  system-\n", program_name);
            fprintf(stderr,"imposed  limit  on  the  maximum number of "\
                        "allowed message queue\n");
            fprintf(stderr,"identifiers system-wide would be exceeded.\n");

         }//if (TROUBLESHOOT==3) {

      } else if (errno == EPERM) {

         snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a EPREM "\
                                          "error", program_name);
         call_sys_log(textbuffer);

         if (TROUBLESHOOT==3) {

            fprintf(stderr,"%s: The process does not have permission to "\
                        "send the signal  to  any\n", program_name);
            fprintf(stderr,"of the target processes\n");

         }//if (TROUBLESHOOT==3) {

      } else if (errno == ESRCH) {

         snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received a ESRCH"\
                                          " error", program_name);
         call_sys_log(textbuffer);

         if (TROUBLESHOOT==3) {

            fprintf(stderr,"%s: The  pid or process group does not exist."\
                        "  Note that an existing\n", program_name);
            fprintf(stderr,"process might be a zombie, a  process  "\
                        "which  already  committed\n");
            fprintf(stderr,"termination, but has not yet been wait()ed "\
                         "for.\n");

         }//if (TROUBLESHOOT==3) {

      } else {

         snprintf(textbuffer, MAX_MESSAGE, "%s: ERROR: Received an unknown"\
                                          " error from IPC", program_name);
         call_sys_log(textbuffer);

         if (TROUBLESHOOT==3) {

            fprintf(stderr,"%s: Unknown error number, error number ="\
                        " %i\n", program_name, errno);

         }//if (TROUBLESHOOT==3) {

      }//if (errno == EACCES) {

}//void print_IPC_error(char* program_name){
```

```
/*****************************************************************************
*
* Function to compare two in6_addr
*
*****************************************************************************/

int cmpaddr(const struct in6_addr *source,
            const struct in6_addr *destination, const int mask) {

  int   count = ZERO;
  int   type  = ZERO;
  int   max   = ZERO;

  if (mask == 128) {

     max  = 4;
     type = 32;

  } else if ((mask%32) == ZERO) {

     type = 32;
     max  = mask/type;

  } else if ((mask%16) == ZERO) {

     type = 16;
     max  = mask/type;

  } else if ((mask%8) == ZERO) {

     type = 8;
     max  = mask/type;

  } else {

     return BADMASK;

  }//if (mask == 128) {


  if (type == 32) {

     for (count=0;count<max;count++){

        if (ntohl(source->in6_u.u6_addr32[count]) >
            ntohl(destination->in6_u.u6_addr32[count])){

           return 1;


        } else if (ntohl(source->in6_u.u6_addr32[count]) <
                   ntohl(destination->in6_u.u6_addr32[count])){

           return -1;

        }//if (source->in6_u.u6_addr32[count] != destination->in6_u.u6_addr32[c

     }//for (count=0;count<max;count++){

   } else if (type == 16) {

     for (count=0;count<max;count++){

        if (ntohs(source->in6_u.u6_addr16[count]) >
            ntohs(destination->in6_u.u6_addr16[count])){

           return 1;

        } else if (ntohs(source->in6_u.u6_addr16[count]) <
```

```
                  ntohs(destination->in6_u.u6_addr16[count])){

           return -1;

        }//if (source->in6_u.u6_addr16[count] != destination->in6_u.u6_addr16[c

     }//for (count=0;count<max;count++){


   } else {

     for (count=0;count<max;count++){

        if (source->in6_u.u6_addr8[count] >
            destination->in6_u.u6_addr8[count]){

           return 1;

        } else if (source->in6_u.u6_addr8[count] <
                   destination->in6_u.u6_addr8[count]){

           return -1;

        }//if (source->in6_u.u6_addr8[count] != destination->in6_u.u6_addr8[cou

     }//for (count=0;count<max;count++){


   }//if (type == 32) {

  return 0;

}//int cmpaddr(const struct in6_addr *source, const struct in6_addr *destinatio


void call_sys_log(char* in_string){

    int    sendsuccess       = ZERO;    // Tells if we had success sending this
    struct messagebuffer     sendbuffer; // Local copy of the send buffer
    sendbuffer.type = SYSLOG;
    snprintf(sendbuffer.text, MAX_MESSAGE, in_string);
    sendsuccess = msgsnd(queueid, (void *) &sendbuffer,
                         sizeof(sendbuffer.text), IPC_NOWAIT);
    if (TROUBLESHOOT==1){

       fprintf(stderr,"%s\n", sendbuffer.text);

    }//if (TROUBLESHOOT==1){

    if(sendsuccess != ZERO){

        fprintf(stderr,"PROBE UTILS: ERROR: Message send failed.  "\
                       "Error: %i\n", errno);
        fprintf(stderr,"%s\n", sendbuffer.text);

    }//if(sendsuccess != ZERO){

}//void call_sys_log(char* in_string){
```

PROBE_UTILS.H

```
/*****************************************************************************
 * This is the header file for all the utilities needed by the probing engine
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: probe_utils.h
 *
 *****************************************************************************/

#ifndef INCLUSION_GUARD_PROGRAM_PROBE_UTILS
#define INCLUSION_GUARD_PROGRAM_PROBE_UTILS

/*****************************************************************************
 *
 * Define externally available programs
 *
 *****************************************************************************/

   void print_IPC_error(char* program_name);
   void call_sys_log(char* in_string);
   int cmpaddr(const struct in6_addr *source,
               const struct in6_addr *destination, const int mask);
   int calc_p_value(const int hop_length[], const int number_of_values);

#endif //INCLUSION_GUARD_PROGRAM_PROBE_UTILS
```

SYS_LOG.C

```
/*****************************************************************************
 * This is the system logger faciltiy
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: sys_log.c
 *
 *****************************************************************************/
#include <time.h>
#include "probe_main.h"
#define  TROUBLESHOOT     0            // 1 = turn on prints
                                       // 2 = turn on flush to force
                                       //     writing to hard drive, no cache
                                       //     Slows system down but good for
                                       //     troubleshooting

#define  SYSLOGNAMELENGTH 27

void sys_log() {

      /*****************************************************************************
      *
      * Define variables needed by the system logger
      *
      *****************************************************************************/
      struct messagebuffer     recvbuffer;

      // Our flag to exit
      int                      done          = PLZCONTINUE;
      // The value of our receive op!!
      int                      rcvsuccess    = ZERO;
      // Number of character converted by time
      int                      convert       = ZERO;
      // The current time in system format
      time_t                   curtime;
      // The time in appropriate format
      struct tm                *loctime;
      // name of the sys log file
      char                     syslog_filename[SYSLOGNAMELENGTH];
      // Holds time in string format
      char                     temptime[26];
      // Used to hold the current date
      char                     sys_date[DATELENGTH];
      // Used to hold error if fprintf fails
      int                      check         = ZERO;


      signal(SIGQUIT, SIG_IGN);

      /*****************************************************************************
      *
      * Process requests until we are told to stop
      *
      *****************************************************************************/

      // Get the current time.
      curtime = time (NULL);

      //Convert it it to local time representation.
      loctime = localtime (&curtime);
      convert = strftime(sys_date, DATELENGTH, "%d%b%Y",loctime);
      if (convert != (DATELENGTH - 1)) {

         fprintf(stderr,"SYS LOG: PROBLEM STARTING, could not get date for "\
```

```
                "file name\n");

}//if (num != (DATELENGTH - 1)) {
snprintf(syslog_filename, SYSLOGNAMELENGTH,"probe_syslog%s.txt", \
                                          sys_date);

FILE *syslog_stream;

syslog_stream = fopen (syslog_filename, "a");

if (syslog_stream == NULL) {

   fprintf(stderr,"SYS LOG: Sys Log unable to open the log file!\n");
   done = STOP;

} else {

  // Get the current time.
  curtime = time (NULL);

  //Convert it it to local time representation.
  loctime = localtime (&curtime);

  snprintf(temptime, 25, asctime(loctime));
  check = fprintf (syslog_stream, "%s SYSLOG: SYS LOG started\n", \
                                  temptime);
  if (check < ZERO) {

     fprintf (stderr, "%s SYSLOG: SYS LOG started\n", temptime);
     fprintf (stderr, "SYSLOG unable to write to file fprinf returned"\
                      " %i\n", check);

  }//if (TROUBLESHOOT) {
  if (TROUBLESHOOT == 1) {

     fprintf (stderr, "%s SYSLOG: SYS LOG started\n", temptime);
     fflush(syslog_stream);

  }//if (TROUBLESHOOT == 1) {
  if (TROUBLESHOOT == 2) {

     fflush(syslog_stream);

  }//if (TROUBLESHOOT == 2) {

}//if (syslog_stream == NULL) {


while (done == PLZCONTINUE) {

    //Get the message off the queue, block until one arrives
    rcvsuccess = msgrcv(queueid, (void *) &recvbuffer,
                        sizeof(recvbuffer.text), SYSLOG, IPC_WAIT);

    if(rcvsuccess == IPC_ERROR) {

        if (errno == EINVAL) {
           fprintf(stderr,"SYS LOG: Message Receive failed.  "\
                          "errno = %i\n", errno);
           snprintf(temptime, 25, asctime(loctime));
           fprintf(syslog_stream, "%s SYS LOG: Message Receive failed."\
                                  " errno %i\n", temptime, errno);
           fprintf(syslog_stream, "SYSLOG: SYS LOG shutting down!\n");
           done = STOP;

        } else {

           fprintf(stderr,"SYS LOG: Message Receive failed.  errno"\
                          " = %i\n", errno);
```

```
        snprintf(temptime, 25, asctime(loctime));
        fprintf(syslog_stream, "%s SYS LOG: Message Receive failed."\
                               " errno %i\n", temptime, errno);

     }//if (errno == EINVAL) {

} else {

   // Get the current time.
   curtime = time (NULL);

   //Convert it it to local time representation.
   loctime = localtime (&curtime);

   char  tempdate[DATELENGTH];
   convert = strftime(tempdate, DATELENGTH, "%d%b%Y",loctime);
   if (convert != (DATELENGTH - 1)) {

      fprintf(stderr,"SYS LOG: PROBLEM with time, check log\n");
      snprintf(temptime, 25, asctime(loctime));
      fprintf (syslog_stream, "%s SYS LOG:Problem converting "\
                              "time\n", temptime);

   } else {

      if (strncmp(tempdate, sys_date,(DATELENGTH - 1)) != ZERO) {

         snprintf(temptime, 25, asctime(loctime));
         fprintf (syslog_stream, "%s SYS LOG: Sys log changing "\
                                 "files\n", temptime);
         fclose (syslog_stream);
         strncpy(sys_date,tempdate,DATELENGTH);
         snprintf(syslog_filename, SYSLOGNAMELENGTH,\
                  "probe_syslog%s.txt", sys_date);
         syslog_stream = fopen (syslog_filename, "a");
         if (syslog_stream == NULL) {

            fprintf(stderr,"SYS LOG: Sys Log unable to open new "\
                           "log file!\n");
            done = STOP;

         }//if (syslog_stream == NULL) {

      }//if (strncmp(tempdate, date,(DATELENGTH - 1) != ZERO) { //Tim


   }//if (num != (DATELENGTH - 1)) {


   if (strncmp("QUIT", recvbuffer.text, 4)){

      snprintf(temptime, 25, asctime(loctime));
      check = fprintf (syslog_stream, "%s %s\n", temptime, \
                       recvbuffer.text);
      if (check < ZERO) {

         fprintf (stderr, "%s %s\n", temptime, recvbuffer.text);
         fprintf (stderr, "SYSLOG unable to write to file fprintf "\
                          "returned %i\n", check);

      }//if (check < ZERO) {
      if (TROUBLESHOOT == 1) {

         fprintf (stderr, "%s %s\n", temptime, recvbuffer.text);
         fflush(syslog_stream);

      }//if (TROUBLESHOOT == 1) {
      if (TROUBLESHOOT == 2) {
```

```
                  fflush(syslog_stream);

               }//if (TROUBLESHOOT == 2) {

            } else {

               done = STOP;
               snprintf(temptime, 25, asctime(loctime));
               check = fprintf (syslog_stream, "%s SYS LOG: Sys log received"\
                                               " quit command\n", temptime);
               if (check < ZERO) {

                  fprintf (stderr, "%s SYS LOG: Sys log received quit "\
                                   "command\n", temptime);
                  fprintf (stderr, "SYSLOG unable to write to file fprintf"\
                                   " returned %i\n", check);

               }//if (TROUBLESHOOT) {
               if (TROUBLESHOOT == 1) {

                  fprintf (stderr, "%s SYS LOG: Sys log received quit "\
                                   "command\n", temptime);
                  fflush(syslog_stream);

               }//if (TROUBLESHOOT == 1) {
               if (TROUBLESHOOT == 2) {

                  fflush(syslog_stream);

               }//if (TROUBLESHOOT == 2) {

            }//if (!strncmp("QUIT", recvbuffer.text, 4)){

         }//if(rcvsuccess == IPC_ERROR) {

      }//while (done == PLZCONTINUE) {


  // Close stream
  fclose (syslog_stream);


}//void probe_generator() {
```

SYS_LOG.H

```
/*****************************************************************************
 * This is the system logger faciltiy
 *
 *
 * Author: Robert J. Poulin, Capt, USAF
 *
 * Program name: sys_log.c
 *
 *****************************************************************************/

#include "probe_main.h"

#ifndef INCLUSION_GUARD_PROGRAM_SYS_LOG
#define INCLUSION_GUARD_PROGRAM_SYS_LOG

    void sys_log() ;


#endif //INCLUSION_GUARD_PROGRAM_SYS_LOG
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

BIONDI, Philippe. Scapy. 2007 [cited 4/6/2007 2007]. Available from http://www.secdev.org/projects/scapy/ (accessed 4/6/2007).

Casado, Martin. disect2.c. in Northern Arizona University [database online]. 2001 [cited 5/17/2007 2007]. Available from http://www.cet.nau.edu/~mc8/Socket/Tutorials/disect2.c (accessed 5/17/2007).

Cisco security advisory: IPv6 routing header vulnerability. 2007a [cited 4/6/2007 2007]. Available from http://www.cisco.com/warp/public/707/cisco-sa-20070124-IOS-IPv6.shtml (accessed 4/6/2007).

Cisco - understanding and configuring the ip unnumbered command. [cited 4/15/2007 2007]. Available from http://www.cisco.com/warp/public/701/20.html (accessed 4/15/2007).

Clauset, Aaron, and Cristopher Moore. 2003. *Traceroute sampling makes random graphs appear to have power law degree distributions.*

Combs, Gerald. Wireshark: The world's most popular network protocol analyzer. 2007 [cited 5/17/2007 2007]. Available from http://www.wireshark.org/ (accessed 5/17/2007).

Conta, A., and S. Deering. 2006. *Internet control message protocol (ICMPv6) for the internet protocol version 6 (IPv6) specification.* Internet Engineering Task Force.

Convery, Sean, and Miller, Darrin. IPv6 and IPv4 threat comparison and best-practice evaluation (v1.0). in IPv6 Eprints Server [database online]. 2006 Available from http://www.6journal.org/archive/00000202/01/v6-v4-threats.pdf (accessed 12/21/ 2006).

Deering, S., R. Hinden, and A. Conta. 1998. *Internet protocol, version 6 (IPv6) specification.* Internet Engineering Task Force.

d'Itri, Marco. Zebra dump parser. in Italian Linux Society [database online]. 2007 [cited 5/18/2007]. Available from http://www.linux.it/~md/software/zebra-dump-parser.tgz (accessed 5/18/2007).

Donnet, Benoit, Huffaker, Bradley, Friedman, Timur and claffy, kc. Implementation and deployment of a distributed network topology discovery algorithm. in arXiv.org [database online]. 2006 [cited 12/20/2006] Available from http://arxiv.org/pdf/cs.NI/0603062 (accessed 2/25/2007).

Donnet, Benoit, Philippe Raoult, Timur Friedman, and Mark Crovella. 2005. Efficient algorithms for large-scale topology discovery. Paper presented at SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, Banff, Alberta, Canada.

Garrett, Aviva, Gary Drenan, and Cris Morris. 2002. *Juniper networks field guide and reference*. 1st ed. Boston: Addison-Wesley Professional.

Govindan, Ramesh, and Hongsuda Tangmunarunkit. 2000. Heuristics for internet map discovery. Paper presented at IEEE INFOCOM 2000 Available from http://www.isi.edu/div7/publication_files/heuristics.pdf (accessed 2/25/2007).

Malone, David, and Niall Richard Murphy. 2005. *IPv6 network administration* O'Reilly Media, Inc.

Meyer, D. University of Oregon Route Views Project in Advanced Network Technology Center [database online]. 2007 [cited 2/25/2007]. Available from http://www.routeviews.org/ (accessed 2/25/2007).

Popoviciu, Ciprian P., Eric Levy-Abegnoli, and Patrick Grossetete. 2006. *Deploying IPv6 networks* Cisco Press.

Schiffman, Mike. The million packet march. 2007 [cited 5/17/2007 2007]. Available from http://www.packetfactory.net/libnet/ (accessed 5/17/2007).

Shaw, Joseph W. TCPDUMP public repository. 2006 [cited 5/17/2007 2007]. Available from http://www.tcpdump.org/ (accessed 5/17/2007).

Spring, Neil T., Ratul Mahajan, and David Wetherall. 2002. Measuring ISP topologies with rocketfuel. Paper presented at SIGCOMM .

Spring, Neil T., David Wetherall, and Thomas E. Anderson. 2003. Scriptroute: A public internet measurement facility. Paper presented at USENIX Symposium on Internet Technologies and Systems.

Waddington, Daniel G., Fangzhe Chang, Ramesh Viswanathan, and Bin Yao. 2003. Topology discovery for public IPv6 networks. *SIGCOMM Comput.Commun.Rev.* 33, (3): 59-68.

Welcome to RIPE.NET. 2007b [cited 5/18/2007 2007]. Available from http://www.ripe.net/ (accessed 5/18/2007)

Yao, Bin, Ramesh Viswanathan, Fangzhe Chang, and Daniel Waddington. 2003. Topology inference in the presence of anonymous routers. Paper presented at INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Air Force Research Laboratory
   Rome, NY
   Attn: Peter Fitzgerald/IFGA